Antti Salomäki

Varastosovelluksen ohjelmistomigraatio

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Tietotekniikan koulutusohjelma
Insinöörityö
27.4.2011

Metropolia

Tiivistelmä

Tekijä(t) Otsikko Sivumäärä Aika	Antti Salomäki Varastosovelluksen ohjelmistomigraatio 64 sivua + 9 liitettä 27.4.2011
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	ohjelmistotekniikka
Ohjaaja(t)	tuotekehitysinsinööri Tapio Puputti lehtori Simo Silander

Tämän opinnäytetyön aiheena on varastosovelluksen ohjelmistomigraatio. Opinnäytetyön tavoitteena oli päivittää 1990-luvulla tehdyn varastosovelluksen ohjelmisto nykyaikaisempaan tekniikkaan. Ohjelmiston tekniikkaan tehtävien muutosten lisäksi tavoitteena oli myös päivittää ohjelman käyttöliittymän ulkonäköä samanlaiseksi kuin yrityksen muissa käyttöliittymissä.

Päivitettävä ohjelmisto on osa Konecranes Oy:n asiakkailla toimivaa pystyrullavaraston varastonhallintajärjestelmää. Sen on tarkoitus palvella asiakasta näyttämällä pystyrullavaraston paperirullatietoja ja nosturin diagnostiikkaa. Uusi ohjelmisto toimii samalla tavalla kuin aikaisempi, mutta tämän lisäksi siihen on lisätty uuden tekniikan tarjoamia käyttöliittymäominaisuuksia.

Opinnäytetyössä perehdytään migraatioprosessin eri vaiheisiin, ohjelmointikielten ulkoasuun, käyttöliittymäsuunnitteluun, migraation tuomiin hyötyihin yrityksen näkökulmasta ja tekniikoihin, joilla uusi ohjelmisto toteutetaan. Insinöörityössä käytettiin paljon apuna eri lähdeaineistoja. Kirjojen ja artikkeleiden lisäksi lähdeaineistona käytettiin dokumentteja, joista suurin osa on Internetistä saatuja verkkodokumentteja.

Opinnäytetyö tehtiin yrityksen pyynnöstä osana Konecranes Oy:n tuotekehitysosaston sovelluskehitysprojektia. Yritykselle opinnäytetyön ajankohtaisuus korostuu, koska vanhan ohjelmiston tekniikka vanhentuu ja sen tekninen tuki on jo päättynyt muutamia vuosia sitten. Työn lopputuloksena on, että varastosovelluksen ohjelmistomigraatio saatiin valmiiksi ja sen käyttöliittymä on myös uudistettu. Päivitetty ohjelmisto toimii tulevien järjestelmien pohjana, jota voidaan hyödyntää yrityksen ohjelmistokehityksessä tulevaisuudessa.

Varastosovelluksen kehityksessä käytettiin apuna sovelluskehitysympäristöjä kuten .NET Framework 3.5, Visual Studio 2008 sekä Expression Blend. Tiedon käsittelemiseen käytettiin Microsoft SQL Server 2008 -tietokantarajapintaa.

Avainsanat	migraatioprosessi, tietokantarajapinta, käyttöliittymä, .NET, Visual Studio 2008, MS SQL Server 2008, Epression Blend

Metropolia

Abstract

Author(s) Title Number of Pages Date	Antti Salomäki Software Migration of Warehouse Application 64 pages + 9 appendices 27 April 2011
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Tapio Puputti, R&D Engineer Simo Silander, Senior Lecturer

The topic of this Bachelor's thesis is the software migration of warehouse application. The aim was to upgrade the old warehouse application's software made in 1990s to meet the standards of modern technology. In addition to technical software upgrading, one of the key goals was to update the user interface of the system to look identical with the company's other interfaces.

The migrateable software is a part of Konecranes Ltd's customers operating erect roll storage's warehouse management system for cranes. Its purpose is to serve customers by showing the paper roll information of erect roll storage and crane diagnostics. The new software works in the same way as earlier, but this also adds new technology offered by the user interface features.

This thesis focuses on the different stages of the migration process models, the syntax of programming languages, user interface design, the migration benefits of the company's perspective and the technologies used for implementation. A lot of help from different source materials was used for the thesis, e.g. books, articles and Web-based documents.

This thesis was commissioned by Konecranes Ltd and written in participation with the company´s R&D department´s software development projects. The company emphasizes the significance of the thesis, because the old software is of obsolete technology and the technical support has already ended a few years ago. The main result is that the software migration of the warehouse application was completed and its user interface has also been redesigned. The updated software is used as basis for the future systems that can be utilized in the enterprise's software development in the future.

Warehouse application development is used to help application development environments such as .NET Framework 3.5. Visual Studio 2008 and Expression Blend. Data processing using Microsoft SQL Server 2008 -database interface.

Keywords	migration process, database interface, user interface .NET, Visual Studio 2008, MS SQL Server 2008, Epression Blend

Sisällys

Lyhenteitä ja käsitteitä

.NET FRAMEWORK

Ohjelmointi-ja ajoympäristö Windows-pohjaisille sovelluksille. Se koostuu lukuisista luokkakirjastoista, jotka jakautuvat eri tasoihin.

ADO.NET *ActiveX Data Objects*. Tietokantarajapinta, joka tarjoaa korkeamman tason rajapinnan tiedon käsittelylle. ADO.NET mahdollistaa tiedon täydellisen kontrollin ohjelman ja tietokannan välillä.

ADODB ADO (ActiveX Data Objects) luokkakirjastoon kuuluva tietokantayhteysobjekti.

BMP *Bitmap Image File*. BMP on bittikarttakuvatiedostoformaatti joka on suosittu Windows-käyttöjärjestelmän ohjelmissa.

CLR *Common Language Runtime*. Ajoympäristö .NET-ohjelmille.

COM *Component Object Model*. COM on alustariippumaton binääristen komponenttien järjestelmämalli. Komponentit kommunikoivat muiden oliopohjaisten komponenttien kanssa. Microsoftin tunnetuimmat COM-mallin mukaiset komponentit ovat OLE- ja ActiveX-komponentit.

DATASET Tietokannan hallintaan liittyvä objekti, joka koostuu tietokannasta haetusta tietosisällöstä.

IDE *Integrated Development Environment*. Integroitu kehitysympäristö joka pitää sisällään ajo- , virhetestaus- sekä suunnitteluympäristön.

IL *Intermediate Language*. Eräänlainen välikieli .NET Frameworkia varten. CLR-ajoympäristö osaa tulkita IL-kieltä.

JIT *Just-in-time compilation*. Dynaaminen ohjelmointikielen kääntäjä, joka kääntää IL-kieltä prosessorin ymmärtämää konekieltä ennen ohjelman ajoa.

LINQ *Language Integrated Query*. .NET Frameworkin komponentti, joka mahdollistaa datakyselyt .NET ohjelmointikielessä.

MDI *Multiple Document Interface*. Lomakkeista koostuvan ohjelman käyttöliittymärajapinta.

MODAALINEN IKKUNA (MDI)

Form-objekti. Sovelluksissa esiintyvät lomakkeet. Kun lomake avataan modaalisena ikkunana, se täytyy sulkea ennen kuin kohdistuksen voi siirtää toiseen objektiin.

MODUULI Sovelluslogiikan yksikkö.

NAMESPACE Kokoelma erilaisia luokkia. Esimerkiksi Shapes-namespacen alla on graafisiin muotoihin liittyviä luokkia.

OHJELMISTOMIGRAATIO

Tietotekniikassa tietojen siirtämiseen käytettävä termi, esimerkiksi otettaessa käyttöön ohjelman uudempi versio.

PIKSELI Bittikarttagrafiikan pienin yksittäinen osa. 1 pikseli = 15 twippiä.

QUERY Tietokannan hallintaan liittyvä termi.

REVISIO Ohjelmatuotteeseen tehtyjen muutosten jälkeinen hallinta-alkio.

SVN *Software Versioning.* Alustariippumaton versionhallintajärjestelmä.

TWIP Näytöstä riippuvainen pituutta kuvaava yksikkö. Yksi twip 1/1440 tuumaa.

VARASTONHALLINTAJÄRJESTELMÄ

Warehouse Management System. Järjestelmän avulla hallitaan varaston sisällä olevia tavaroita ja pyritään optimoimaan ja tehostamaan kaikkia varaston sisällä tehtyjä prosesseja. Järjestelmä koostuu eri komponenteista joihin kuuluu muun muassa sovelluksia, väyläratkaisuja ja tietokanta.

VARASTOSOVELLUS

Varastosovellus on osa pystyrullavaraston varastonhallintajärjestelmää. Sen avulla voidaan siirtää tavaroita, hyllyttää ja seurata varastonosturin diagnostiikkaa. Varastosovellus koostuu käyttöliittymästä ja businesslogiikasta.

VB.NET *Visual Basic.NET*. Ohjelmointikieli.

VB6 *Visual Basic 6*. Ohjelmointikielen 6:s versio.

WPF *Windows Presentation Foundation.* Yksi .NET Framework -sovelluskehyksen graafinen ali-järjestelmä, jonka avulla voidaan luoda Windows-pohjaisia sovelluksia. WPF-sovelluksissa käytetään DirectX-grafiikkamoottoria käyttöliittymien mallintamiseen.

XAML *eXtensible Application Markup Language.* Deklaratiivinen merkintäkieli, jota käytetään .NET-pohjaisten sovellusten käyttöliittymissä.

1 Johdanto

Tämä insinöörityö käsittelee varastosovelluksen ohjelmistomigraatiota .NET-tekniikkaan. Työssä tutustutaan päivitetyn ja päivitettävän ohjelmistoprojektin arkkitehtuuriin, ohjelmointikielten ulkoasuun, ohjelmistomigraation tuomiin hyötyihin, sekä uuden ohjelmiston suunnitteluun ja sen toteutukseen.

Opinnäytetyön tilaajana toimi Konecranes Oy, joka on yksi maailman johtavista nostolaitevalmistajista. Yritys toimittaa tuottavuutta lisääviä nostolaiteratkaisuja ja palveluita prosessiteollisuudelle, ydinvoimaloille, laivanrakennusteollisuudelle ja satamille.

Käännettävä ohjelmisto on osa yrityksen tuotekehitysosaston kehittelemää pystyrullavaraston varastonhallintajärjestelmää. Varastonhallintajärjestelmä on ollut käytössä Konecranes Oy:n asiakkailla jo useamman vuoden. Asiakkaina ovat yritykset, jotka toimivat paperi- ja metalliteollisuudessa.

Aiempi varastosovelluksen ohjelmisto on toteutettu Visual Basic 6 -tekniikalla. Käännösprosessin tavoitteena on kääntää vanha ohjelmisto Visual Basic .NET -versioksi käyttäen hyväksi .NET Framework 3.5 -komponenttikirjastoa. Raportissa esitellään myös .NET-tekniikan yhteensopivuutta eri ohjelmistokehitysalustoihin.

Varastosovellus on Server-client tyyppinen ohjelmisto, jonka avulla asiakas voi seurata reaaliajassa pystyrullavarastossa olevien paperirullien paikka- ja spesifikaatiotietoja. Visual Basic 6 -tekniikan vanhentumisen, sekä sen teknisen tuen loppumisen vuoksi ohjelmistomigraatio osoittautui tärkeäksi projektiksi yrityksessä.

Varastosovelluksen VB.NET-versiossa on samat toiminnot kuin vanhassakin ja palvelee asiakasta samoin kuin edeltäjänsä, mutta käyttöliittymän ulkoasu on uudenaikainen ja sen käytettävyyteen on lisätty muutamia lisäominaisuuksia. Tämän lisäksi uuteen versioon on hyödynnetty .NET Framework 3.5 -komponenttikirjaston vektorigrafiikkaan perustuvia näyttöriippumattomia käyttöliittymäkomponentteja.

Varastosovelluksen VB.NET-version toteutukseen käytettiin Visual Studio 2008 -kehitysympäristöä ja käyttöliittymän ulkoasun suunnitteluun Expression Blend piirtotyökalua. Uusien sovellusjärjestelmien kehittäminen ja uusien tekniikoiden soveltaminen uusimmilla alustoilla tuo yritykselle lisää haasteita ohjelmistokehitykseen. .NET-tekniikan monipuolisuuden sekä uusien ominaisuuksien myötä ohjelmistotuotteeseen saadaan lisä-arvoa, ja sen elinkaari kasvaa.

2 Työn lähtökohdat ja tavoitteet

2.1 Ohjelmistomigraation tavoitteet

Ohjelmistomigraation tavoitteena on toteuttaa uusi versio vanhasta Visual Basic 6 -pohjaisesta ohjelmistosta VB.NET-pohjaiseksi sovellusjärjestelmäksi. Ohjelmistomigraatiota ei voida suoranaisesti pitää ohjelmistopäivityksenä, sillä varastosovelluksen VB.NET-versiossa on samat toiminnot kuin vanhassakin ja varastosovelluksen arkkitehtuuri on pysynyt lähes muuttumattomana. Varastosovelluksen VB.NET-versioon on lisätty muutamia lisä-ominaisuuksia ja varastosovelluksen käyttöliittymä on uudenaikainen verrattuna vanhaan.

VB.NET-pohjaisen varastosovelluksen kehityksessä hyödynnettiin Microsoftin kehittelemiä .NET Frameworkin kanssa yhteensopivia ohjelmistokehitysalustoja kuten Visual Studio 2008 -sovelluskehitinympäristöä, käyttöliittymäsuunnitteluun ja vektorigrafiikkaan Expression Blend:ä ja tiedonhallintaan MS SQL Server:ä. Kehitysalustojen yhteensopivuuden vuoksi .NET-pohjaisten järjestelmien kehittäminen on tehokasta. Näiden kehitysalustojen tukemana ohjelmistomigraation yhtenä tavoitteena on saada ohjelmiston käyttöliittymästä ja kokonaisuudesta uudenaikaisempi tuote, joka tuo lisä-arvoa yrityksen ohjelmistotuotteelle.

Yhtenä ohjelmistomigraation tavoitteista voidaan pitää myös varastosovelluksen VB.NET-pohjaisen version käyttöohjeiden laatimista, joka on tärkeä osa ohjelmistotuotantoa. Hyvät käyttöohjeet antavat loppukäyttäjälle helpon lähestymistavan uuteen VB.NET-pohjaiseen sovellukseen. Ohjelmistomigraatiota tehtäessä täytyy myös dokumentoida työn tuloksia säännöllisin väliajoin ja ylläpitää projektin versionhallintaa.

2.2 Rajaukset

Tässä insinöörityössä keskitytään nosturin varastosovelluksen ohjelmistomigraatioon ja ohjelman käyttöliittymään tehtyihin muutoksiin. Ohjelmistomigraatio rajoittuu varastosovelluksen kahteen eri versioon: Visual Basic 6:een ja Visual Basic .NET:iin sekä VB6- ja VB.NET-ohjelmointikieliin ja niiden välisiin ulkoasueroihin. Opinnäytetyössä esitellään myös muutamia ohjelmointiesimerkkejä, jotka liittyvät .NET-pohjaisen varastosovelluksen käyttöliittymäominaisuuksiin.

2.3 Ohjelmistomigraation tuomat hyödyt yrityksessä

Monet yritykset ovat tehneet tietojärjestelmiä Visual Basic 6 -ohjelmointikielellä käyttäen Visual Basic -kehitysympäristöä sen suunnittelussa. Visual Basic 6 -versio julkaistiin vuonna 1998 ja on edelleen laajalti käytössä. Viime vuosina ohjelmistokehitys on muuttunut paljon ja uusia tekniikoita sovelletaan mitä ihmeellisimpiin järjestelmiin, ja näin ollen monet yritykset ovat päivittäneet ohjelmistojaan .NET-tekniikkaan yhteensopivuuden ja sen tuomien lisäominaisuuksien takia. [1.]

Miksi yritykset sitten päivittävät tietojärjestelmiään .NET-pohjaisiksi?

- Jos yrityksessä kehitetään Web-pohjaisia sovelluksia, niin .NET tarjoaa tähän huomattavan parannuksen ASP.NET-rajapinnan avulla.
- .NET:n avulla ohjelmistokehittäjien tuotettavuus paranee ja sovelluksiin saadaan entistä enemmän monipuolisuutta .NET Framework -ohjelmistokehyksen sekä VB.NET-ohjelmointikielen helposti ymmärrettävän ulkoasun ansiosta.
- Vahvistaa ohjelmistotuotteen laatua sekä arvoa.
- Kustannukset vähenevät liiketoiminnassa.
- Visual Basic 6 -version tekninen tuki loppui vuonna 2008, joten VB.NET:n käyttäjille on käytössä ympärivuorokautinen ohjelmistotuki.
- Ohjelmiston ylläpito paranee. Tämä ilmenee siten, että .NET tarjoaa vektorigrafiikan tuottamiselle tekniikan, jota on helppo ylläpitää .NET Framework -ohjelmistokehystä tukevien ohjelmistokehitysalustojen ansiosta. [3, s. 1 - 23.]

Yrityksen VB6-sovelluksen päivittäminen .NET-pohjaiseksi sovellukseksi ei välttämättä ole kuitenkaan ihan helppoa, sillä ohjelmistomigraatio voi olla pitkä prosessi ohjelmistokehityksessä. Yrityksen työntekijät ovat mahdollisesti työskennelleet vain Visual Basic 6:n parissa vuosikaudet eivätkä välttämättä tiedä .NET-ohjelmoinnista mitään. Monissa yrityksissä käytetään silti Visual Basic 6:ta, sillä se toimii kuitenkin hyvin vanhoissa Windows-käyttöjärjestelmissä.

Jos yritys kuitenkin haluaa päivittää VB6-sovelluksia, niin siihen löytyy useita menetelmiä. Yksi hyvä menetelmä on palkata yritykseen .NET-kehittäjä tekemään Visual Basic 6 -ohjelmistomigraatio VB.NET:iin tai kouluttaa jo vanha Visual Basic 6 -kehittäjä käyttämään VB.NET:ä.

Yritys voi vaihtoehtoisesti myös palkata konsultin, joka opastaa yrityksen Visual Basic 6 -kehittäjiä uuteen VB.NET-ohjelmointikieleen ja .NET-ympäristöön. Nämä palvelut voivat kuitenkin tulla yritykselle kalliiksi ja viedä paljon aikaa. Hyvä ja suhteellisen halpa tapa on palkata yritykseen .NET-kehittäjä, joka tuntee uuden tekniikan parhaiten sekä voi tarvittaessa opastaa työkavereitaan uusissa asioissa.

Yksi hyvä syy ohjelmistomigraatiolle on tietokoneprosessorien tehokkuuden kasvu. Uusien tekniikoiden tuomat lisäominaisuudet tarvitsevat koneelta aiempaa enemmän laskentatehoa ja eritoten grafiikkamoottori, joka vastaa ohjelmistossa esiintyvän grafiikan tuottamisesta. WPF-rajapinnan tuomia graafisia ominaisuuksia on hyödynnetty varastosovelluksen VB.NET-versioon.

Sellaisessa yrityksessä, jossa tietokoneet toimivat suhteellisen nykyaikaisella tasolla, ohjelmistojakin tulisi päivittää. Ohjelmistojen sekä tietojärjestelmien tekniikka tulee olla samalla tasolla kuin tietokoneen antaman laskentatehokkuuden, jotta saadaan optimoitua paras hyöty ohjelmistosta.

Yritykset, jotka osaksi toimivat IT-sektorilla ja ovat kehitelleet omia tietojärjestelmiään omiin tuotantoprosesseihin, ovat kilpailukyvyltään muita yrityksiä edellä ohjelmistotuotteeseensa asetetun brändin, imagon ja maineen vuoksi. Yritykset, jotka kehittävät Windows-pohjaisia sovelluksia, tarvitsevat aina .NET-kehittäjiä. Jos yrityksen sisällä on jo omaa IT-alan osaamista, niin silloin voidaan säästää kustannuksissa ohjelmistomi-

graatiota toteuttaessa. Nämä menot näkyvät parhaiten ulkoistumistoiminnassa eli yritys ostaa ohjelmistotuotteen tai palvelun toiselta yritykseltä. [4.]

3 Käytetyt työkalut ja menetelmät

Varastosovellus on alun perin toteutettu Visual Basic -kehitysympäristöllä (IDE). Kehitysympäristön käyttöliittymä on jo hieman vanhanaikainen verrattuna .NET-kehityksessä käytettävään Visual Studio -kehitysympäristöön. Visual Basic -kehitysympäristön tekninen tuki on päättynyt muutamia vuosia sitten. Tämä on myös oleellinen syyn varastosovelluksen päivittämiselle.

3.1 Visual Basic 6 -taustaa

Visual Basic 6 on Microsoftin kehittämä ohjelmointikielistandardi vuodelta 1998. Kieli on integroitu kehitysympäristöön Visual Basic (IDE).

Kieli on periytynyt Visual Basicista, jonka ensimmäinen versio VB 1.0 otettiin käyttöön vuonna 1991. Visual Basic sen sijaan on sukua BASIC-ohjelmointikielelle. Sukuisuus näkyy kielen syntaksissa, joka on säilynyt melkein samanlaisena VB.NET-ohjelmointikieleen saakka. [2, s. 3.]

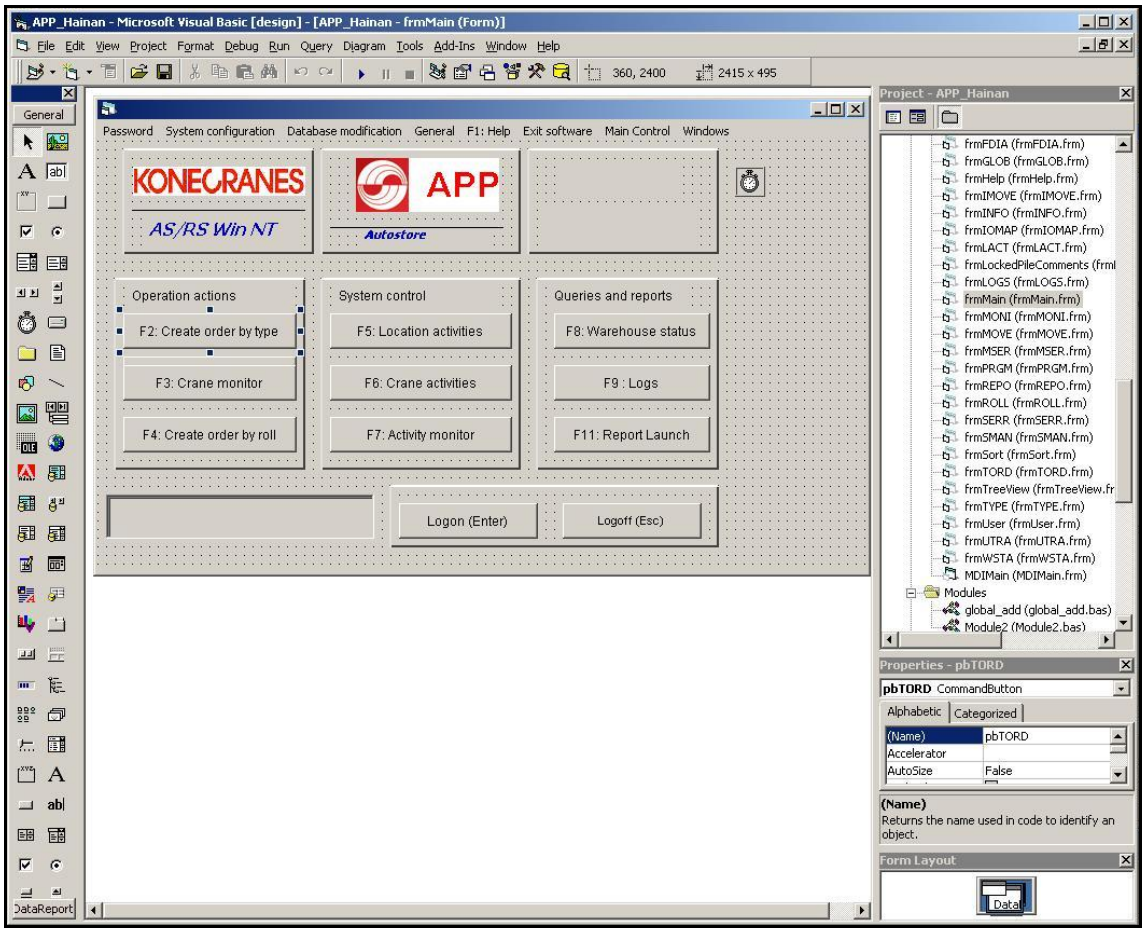
3.2 Visual Basic -kehitysympäristö (IDE)

Visual Basic -kehitysympäristöllä voidaan toteuttaa Windows-sovelluksia. IDE tulee sanoista Integrated Developing Environment, sillä kehitysympäristössä yhdistyvät monet toiminnot, kuten suunnittelu, ohjelmointikirjoitus, käännös ja virheidenetsintä.

Visual Basic -kehitysympäristölle on ominaista sen Drag & Drop -tyyppinen suunnittelu. Siinä ohjelmoija yksinkertaisesti vain raahaa haluamansa komponentin komponenttilistalta sovelluksessa käytettävään graafiseen käyttöliittymään. Tämän tekniikan on kehittänyt Alan Cooper ja hänen yrityksensä Tripod. Microsoft ryhtyi yhteistyöhön Cooperin kanssa, ja he alkoivat kehitellä ohjelmoitavia graafisia lomakkeita (Form) Windows 3.0

-käyttöjärjestelmään. Tätä tekniikkaa käytetään myös myöhemmin ilmestyneessä Visual Studiossa. [1.]

Kehitysympäristö koostuu (kuva 1) mm. elementeistä MenuBar (valikkorivi), Context Menus (kontekstivalikot), Toolbars (työkalurivit), Toolbox (työkaluruutu), Project Explorer Window (Project-ikkuna), Properties Window (Properties-ikkuna), Object Browser (objectien selain), Form Designer (lomakesuunnittelu) ja Code Editor Window (Code-ikkuna) [2, s. 15 - 17.]



Kuva 1. Visual Basic -kehitysympäristö

3.3 .NET Framework ja kehitysalustat

.NET Framework on Microsoftin kehittämä ohjelmistokomponenttikirjasto, jota Visual Studio.NET -kehitysympäristö käyttää. Se tukee useita ohjelmointikieliä ja mahdollistaa kielten yhteensopivuuden. .NET Frameworkin ajoympäristönä toimii CLR. Sen avulla ohjelmointikoodia voidaan kirjoittaa käyttäen useampaa .NET-perheen ohjelmointikieltä

samanaikaisesti, sillä CLR on osittain kieliriippumaton ajoympäristö niin sanottu cross-language. Näitä kieliä ovat muun muassa Visual Basic.NET (VB.NET), C#, C++ ja Jscript.NET.

Komponenttikirjasto pitää sisällään BCL:n (Base Class Libraryn), joka tarjoaa laajan valikoiman uusia ohjelmistokehitykseen liittyviä ominaisuuksia. Näitä ovat mm. käyttöliittymä-, tietokantayhteensopivuus-, salaus-, web-sovellus- sekä verkkoviestintäkomponentit. Näitä komponentteja käyttämällä ja yhdistelemällä voidaan luoda kehittynyt ja uudenaikainen tietojärjestelmä ohjelmistomigraatiolla. [6; 7.]

.NET Framework:n arkkitehtuuri koostuu kahdesta osasta: luokkakirjastoista sekä CLR -ajoympäristöstä. CLR:n toiminta perustuu ohjelman koodin kääntämiseen esikäännettyyn muotoon IL-kieleksi (Intermediate Language) ja tämän jälkeen binäärimuotoon, jota käyttöjärjestelmä voi lukea ja suorittaa. Sen päätoimiset tehtävät ovat käännetyn ohjelman ajaminen, ohjelmasäikeiden hallinta, ajonaikainen turvallisuus ja käyttövarmuus. Käännös binäärimuotoon tapahtuu JIT-kääntäjän avulla (Just In Time). Tämän mahdollistaa optimoinnin eri keskusprosessorityypeille kuten esimerkiksi x86:lle. [8.]

.NET Framework ei ole alustariippumaton (Cross-platform) vaan vaatii toimiakseen Microsoftin käyttöjärjestelmän. Käyttöjärjestelmät kuten Windows XP, Windows 2000, Windows 2003, Windows Vista ja Windows 7 tukevat .NET Framework:a. Novell nimisen ohjelmistoyrityksen tuotekehitysyksikkö on kuitenkin luvannut laajentaa .NET Frameworkin toimivuutta muihinkin alustoihin kuten Linux tai Mac OS X. Projekti tunnetaan nimellä ”Mono” ja sen ensimmäinen vakaa versio on julkaistu vuonna 2004. [8.]

3.3.1 .NET Framework -versiot

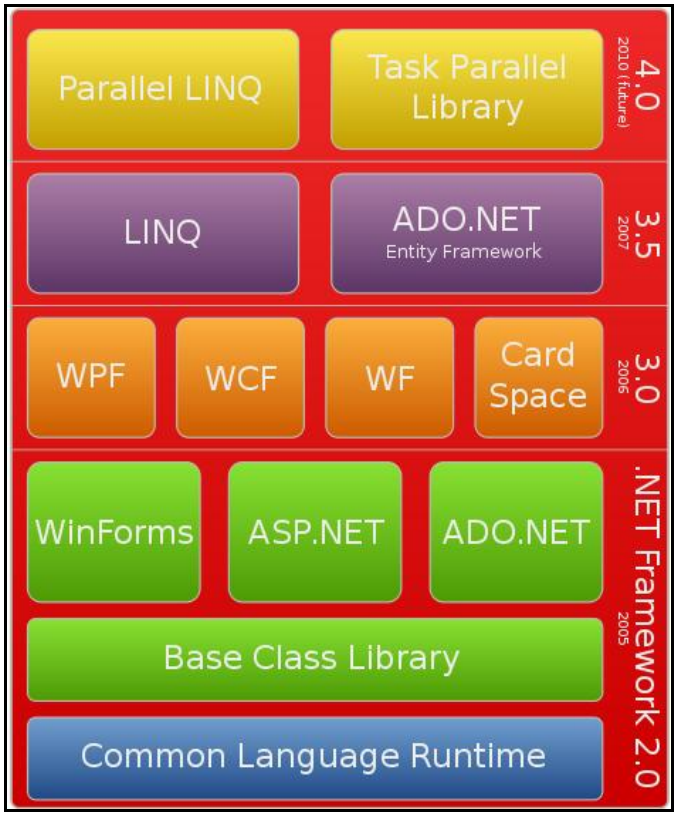
.NET-versioiden kehittäminen alkoi 1990-luvun lopulla alun perin nimellä Next Generation Windows Services (NGWS). Vuonna 2000 julkaistiin ensimmäiset beta-versiot nimellä .NET 1.0. Versio .NET Framework 3.0 on mukana Windows Server 2008 - ja Windows Vista - käyttöjärjestelmissä. Tämän jälkeen ilmestyi .NET Framework 3.5, joka on mukana Windows 7 -käyttöjärjestelmässä. Uusin versio 4.0 ilmestyi 12. huhtikuuta 2010, ja se julkaistiin yhdessä Visual Studio 2010 -kehitysalustan kanssa (taulukko 1). .NET Framework -perheeseen kuuluu myös kaksi muuta versiota, jotka on tar-

koitettu sulautettuihin laitteisiin. Näitä ovat .NET Compact Framework, joka on suunniteltu Windows CE -alustalla toimiville Windows-laitteille sekä .NET Framework Micro, joka on enemmän suunnattu huomattavasti vähemmän laskentatehoa tarvitseville laitteille. [8.]

Taulukko 1. .NET-versioiden julkaisemisajankohdat.

Versio	Version numero	Julkaisupäivä	Visual Studio	Oletus Windows
1.0	1.0.3705.0	2002-02-13	Visual Studio . NET	
1.1	1.1.4322.573	2003-04-24	Visual Studio. NET 2003	Windows Server 2003
2.0	2.0.50727.42	2005-11-07	Visual Studio 2005	Windows Server 2003 R2
3.0	3.0.4506.30	2006-11-06		Windows Vista, Windows Server 2008
3.5	3.5.21022.8	2007-11-19	Visual Studio 2008	Windows 7, Windows Server 2008 R2
4.0	4.0.30319.1	2010-04-12	Visual Studio 2010	

Ohjelmistomigraatiossa käytetään .NET Frameworkin versiota 3.5 ja ohjelmointirajapintana WPF:ää, joka julkaistiin vuonna 2006. Ohjelmointikielenä käytetään Visual Basic.NET:ä (VB.NET). Versio 3.5 tarjoaa WPF:n lisäksi uusia rajapintoja tietokannan käsittelylle muun muassa LINQ ja ADO.NET Entity Framework (kuva 2). Molempia hyödynnetään varastosovelluksen .NET-version kehityksessä. [8.]



Kuva 2. .NET Framework -arkkitehtuuri

3.3.2 Visual Studio 2008

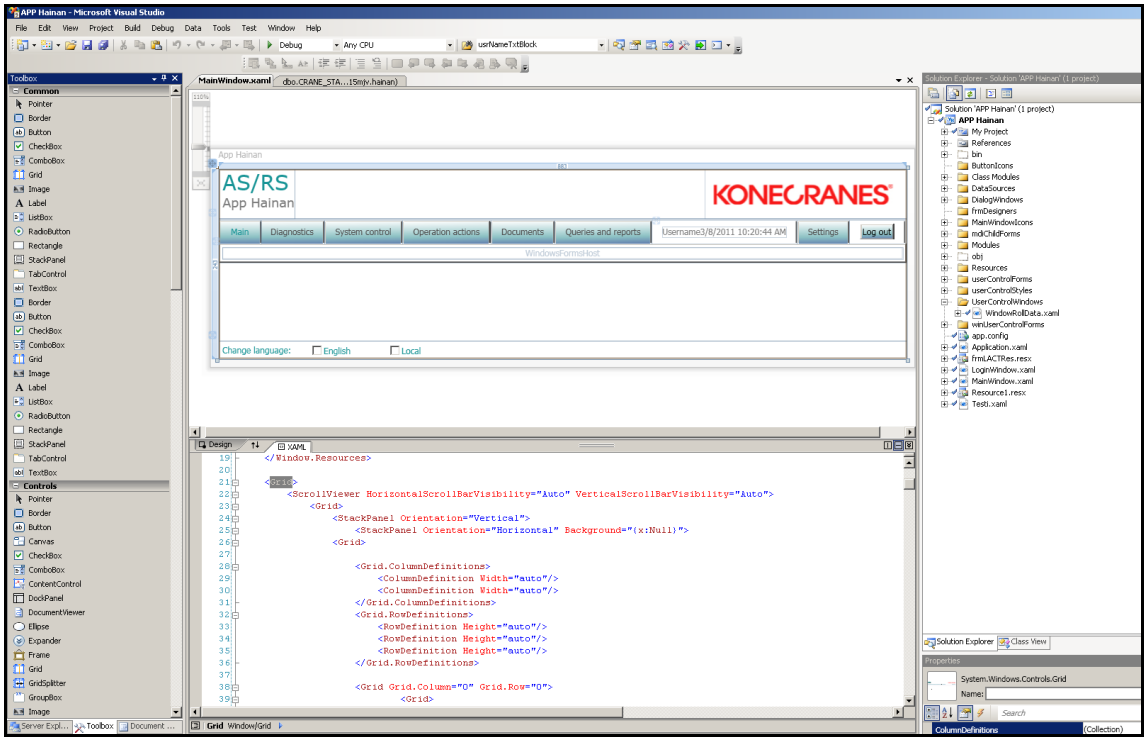
Visual Studio 2008 on ohjelmankehitysympäristö, jonka avulla voidaan tehdä Windows- web- ja mobiilisovelluksia eri tarkoituksiin. Siinä voidaan käyttää useita .NET-arkkitehtuuriin perustuvia kieliä kuten Visual Basicia, C++:aa, C#:aa ja J#:aa. Visual Studion avulla on helppoa tehdä graafisia käyttöliittymiä. Ehkä juuri tämän vuoksi se on niin suosittu kehitysalusta suunnittelijoiden keskuudessa. Vuonna 2002 Microsoft julkaisi Windows XP -alustalle Visual Studio .NET -version.

Kehitysalustaan voidaan liittää myös erilaisia täydennysohjelmia kuten Intel Visual Fortran ja Visual SVN. Visual SVN on Visual Studion liitännäinen, jonka avulla voidaan ylläpitää ohjelmiston versionhallintaa.

.NET-tekniikan ilmestymisen myötä Visual Studion kielet voidaan tulkata yhteiselle kielelle, jota kutsutaan nimellä CLR, joka on lyhenne sanoista Common Language Runti-

me. Toimiakseen se tarvitsee vähintään .NET Framework -kirjaston version 1.0 koneelle, jossa ohjelma ajetaan.

Visual Studio 2008 kehitysalustana on varsin monipuolinen ohjelmistotuotannossa. Siinä on tarvittavat toiminnot hyvin esillä ja käyttäjä voi muokata käyttöliittymän ulkoasua mielensä mukaan. Vasemmassa reunassa näkyvä Toolbox koostuu käyttöliittymä kontrolleista ja yläreunassa sijaitsevat valikot ja menu-paneeli. Alareunassa on käyttöliittymän lähdekoodi (XAML). Oikeassa reunassa on projektin puurakenne. Projektin rakenne koostuu graafisista ikkunoista, tyylisivuista, moduuleista ja projektiin liitetyistä ulkoisista resursseista, joita ovat muun muassa kuvat. Keskellä on esitelty varastosovelluksen .NET-version pääsivun käyttöliittymä (kuva 3). [9.]



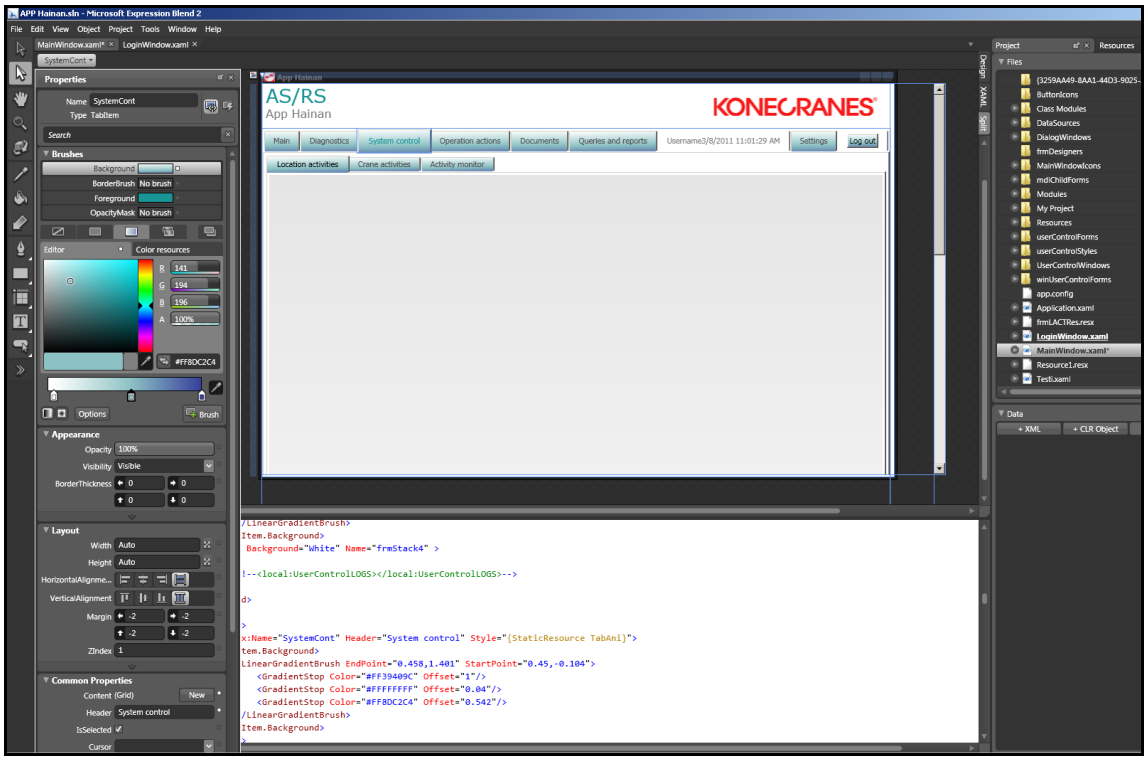
Kuva 3. Visual Studio 2008

3.3.3 Expression Blend

Expression Blend on Microsoftin kehittämä graafinen käyttöliittymäsuunnitteluun tarkoitettu kehitysympäristö. Expression Blend generoi XAML-kieltä eli kun käyttäjä piirtää ohjelmassa jotain, niin piirretty kuvio generoituu XAML-kieleksi.

Expression Blend 2.0 tukee Microsoft Silverlight rajapintaa web-sovelluksissa. Se tarjoaa paljon erilaisia toimintoja kuten animaatioita, vektorigrafiikkaa, interaktiivisen vuorovaikutuksen käyttäjän kanssa ja videoiden toistamisen. Osa varastosovelluksen käyttöliittymän grafiikasta voidaan toteuttaa uudelleen Expression Blend -kehitysympäristöllä.

Työkalu ei ole sen erikoisempi kuin tavanomaiset piirtotyökalut, kuten esim. MS Paint tai Adobe PhotoShop. Esillä on vain tärkeimmät toiminnot, joiden avulla käyttäjä voi luoda nopeasti vektorigrafiikalla tuotettuja komponentteja. Expression Blendin käyttöliittymä on melko samanlainen kuin Visual Studion: esimerkiksi projektipuu ja käyttöliittymän lähdekoodi on samassa paikassa. Vasemmassa reunassa on Properties-paneeli, josta käyttöliittymäkomponentin ominaisuuksia voidaan muokata (kuva 4). [10.]



Kuva 4. Expression Blend 2

3.3.4 Microsoft SQL Server

Microsoft SQL Server on tiedonhallintaan käytettävä työkalu. Työssä on käytetty versiota Microsoft SQL Server 2008. Sen avulla yrityksen tieto- ja analysointisovellusten turvallisuus, skaalautuvuus ja käytettävyys ovat aikaisempia tietokantajärjestelmiä parempia.

Sovellusten luonti, käyttönotto ja hallinta on helppoa SQL Serverin tarjoamien työvälineiden ansiosta. Varastosovelluksen .NET-pohjaiseen versioon hyödynnetään ADO.NET-rajapinnan tarjoamia komponentteja Visual Studion sekä SQL Serverin välillä. SQL Server 2005 ja siitä uudemmat päivitykset integroituvat hyvin Visual Studio 2008 -ohjelmistokehitysalustan kanssa. [11, s. 9 - 34.]

SQL Serverin avulla yritys voi hyödyntää Business Intelligence -ratkaisuja. Tämä tarkoittaa sitä, että kaikilla tietokantaa käyttävillä työntekijöillä on mahdollisuus hyödyntää BI:n tarjoamia ratkaisuja. Näitä ovat muun muassa kattavuus, turvallisuus, integraatio muihin järjestelmiin. Arkkitehtuuri jakautuu SQL Serveriä tukevien rajapintojen sisään (kuva 5). [11, s. 2040 - 2044.]

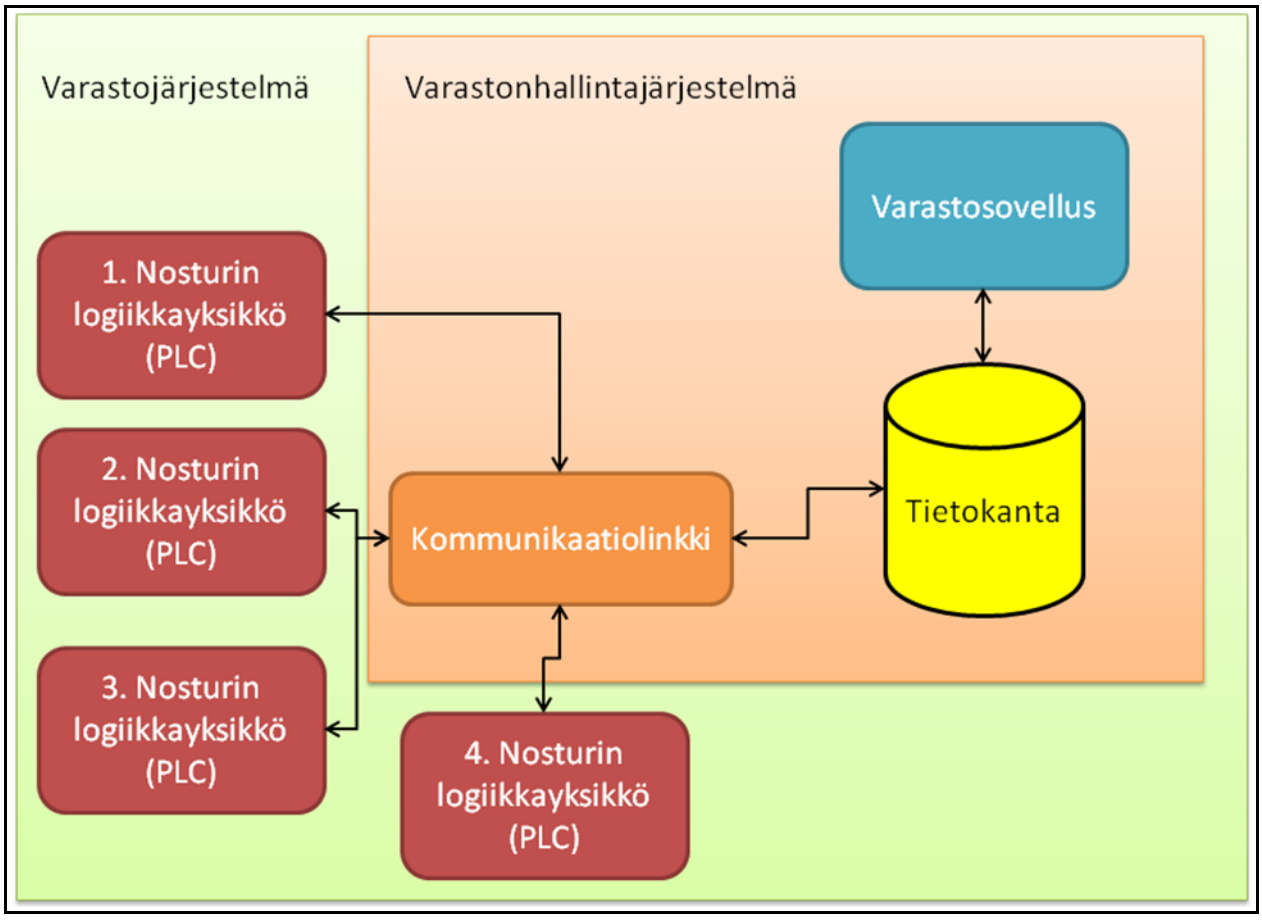
Kuva 5. SQL Server 2008 -ydinkomponentit

4 Varastosovelluksen arkkitehtuuri ja käyttöliittymä

4.1 Yleisarkkitehtuuri

Varastosovellus on osa pystyrullavaraston varastonhallintajärjestelmää. Varastosovellus voidaan määrittää yhtenä komponenttina varastonhallintajärjestelmän toimintaa kuvaavassa yleisarkkitehtuurissa.

Varastojärjestelmä koostuu järjestelmää hallinnoivista kokonaisuuksista ja komponenteista. Näihin kuuluu muun muassa järjestelmää tukevia sovelluksia, varastosovellus, varastonosturin logiikka yksikkö (PLC), tietokanta ja väyläratkaisut. Yleisarkkitehtuuri on rajattu vain varastosovellukseen liittyviin rajapintoihin (kuva 6).

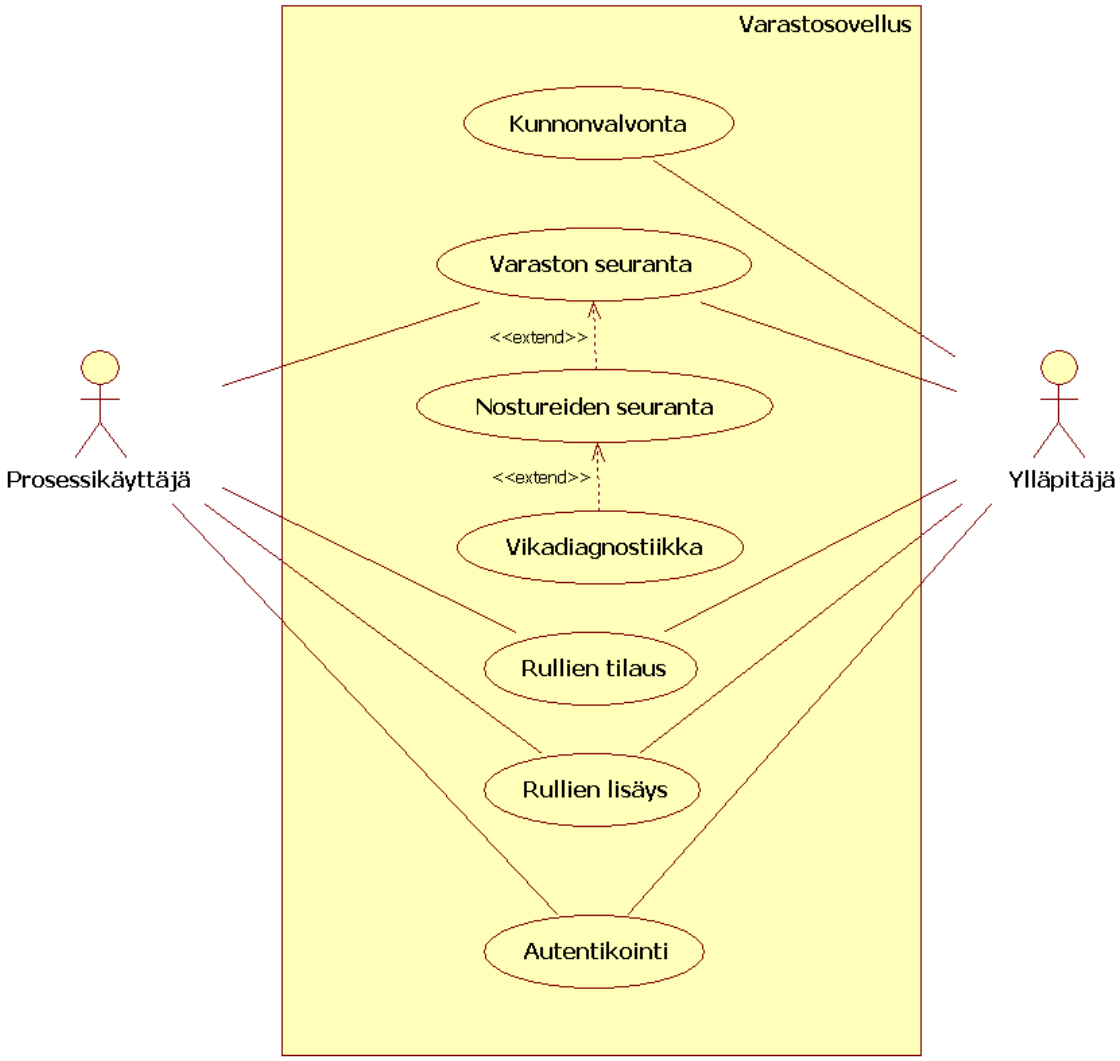


Kuva 6. Yleisarkkitehtuuri

4.2 Varastosovelluksen käyttötapauskaavio

Varastosovelluksen käyttötapaukset jakautuvat kahteen käyttäjäryhmään; prosessikäyttäjä ja ylläpitäjä. Ylläpitäjä huolehtii sovelluksen kunnonvalvonnasta ja vastaa sovelluksen toimivuudesta. Ylläpitäjä voi lisätä uusia käyttäjiä tai muokata käyttäjien oikeuksia, mutta nämä muutokset tehdään suoraan tietokantaan mitä varastosovellus käyttää. Ylläpitäjällä on omien oikeuksien lisäksi samat oikeudet kuin prosessikäyttäjällä.

Prosessikäyttäjä voi seurata sovelluksen avulla pystyrullavaraston statistiikkaa, johon liittyvät nosturin seuranta ja vikadiagnostiikka. Prosessikäyttäjä voi tilata rullia tai ohjata niitä uuteen paikkaan. Tämän lisäksi prosessikäyttäjä voi lisätä rullia sovelluksen avulla tietokantaan. Molemmilta käyttäjäryhmiltä vaaditaan autentikointia sovellukseen sisäänkirjautuessa (kuva 7).



Kuva 7. Varastosovelluksen käyttötapauskaavio

4.3 Sovellusarkkitehtuuri (VB6)

Päivitettävän ohjelmiston sovellusarkkitehtuuri muodostuu viidestä eri kerroksesta (kuva 8).

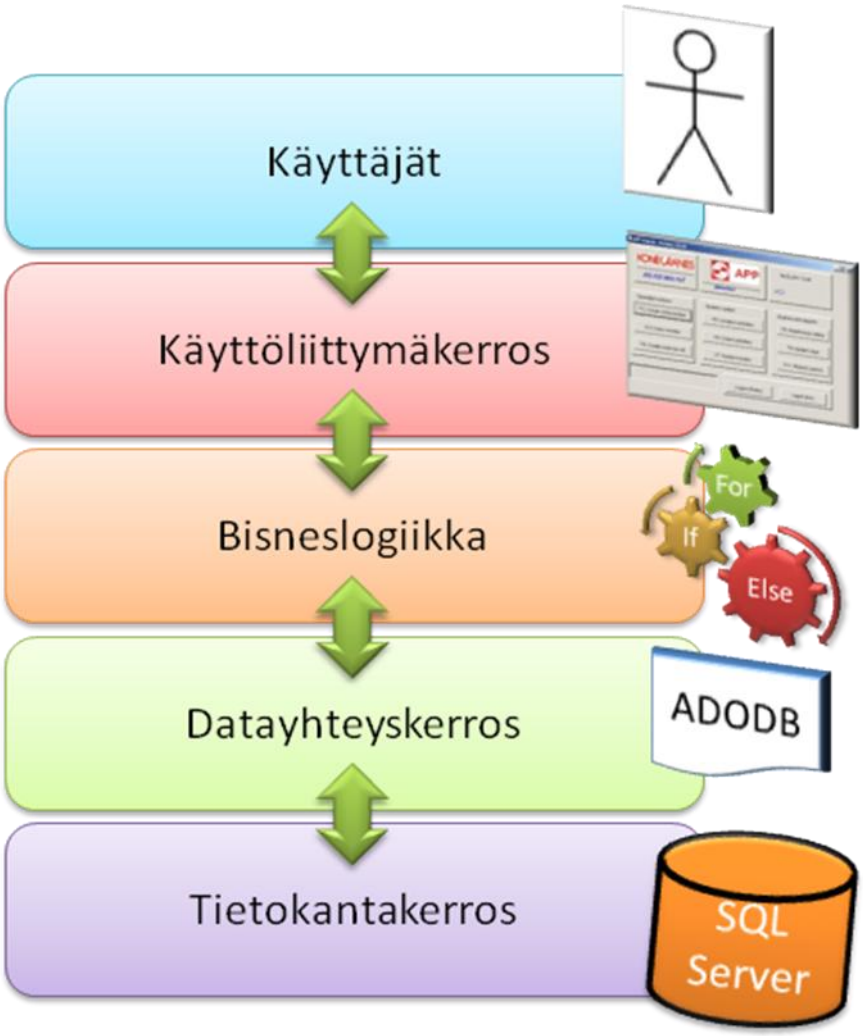
1) Käyttäjät käyttävät sovellusta. Pääsääntöisesti käyttäjänä toimii ihminen.

2) Käyttöliittymäkerros tarjoaa näkymän sovelluksesta käyttäjälle. Se kommunikoi sovelluksen logiikan kanssa ja prosessoi käyttäjän syötteitä.

3) Bisneslogiikka pitää sisällään sovelluksen logiikan. Käsittelee siirrettävää dataa operaatioille asetettujen ehtojen mukaisesti. Bisneslogiikka vastaa myös osaksi käyttöliittymän toiminnallisuuksista esimerkiksi käyttöliittymässä esiintyvät animaatiot.

4) Datayhteyskerros tarjoaa komponentit tiedon käsittelylle, jotka sisältävät tiedon sisällön ja rakenteen. Kommunikoi tietokannan kanssa. Tiedon käsittelemiseen käytetään esimerkiksi ADODB- ja DataSet-objekteja.

5) Tietokanta tarjoaa rajapinnan sovelluksessa käytettävälle tiedolle. Datayhteyteen käytettävät komponentit ovat liitoksissa tähän kerrokseen. Tietokantarajapintana on SQL Server. [5.]

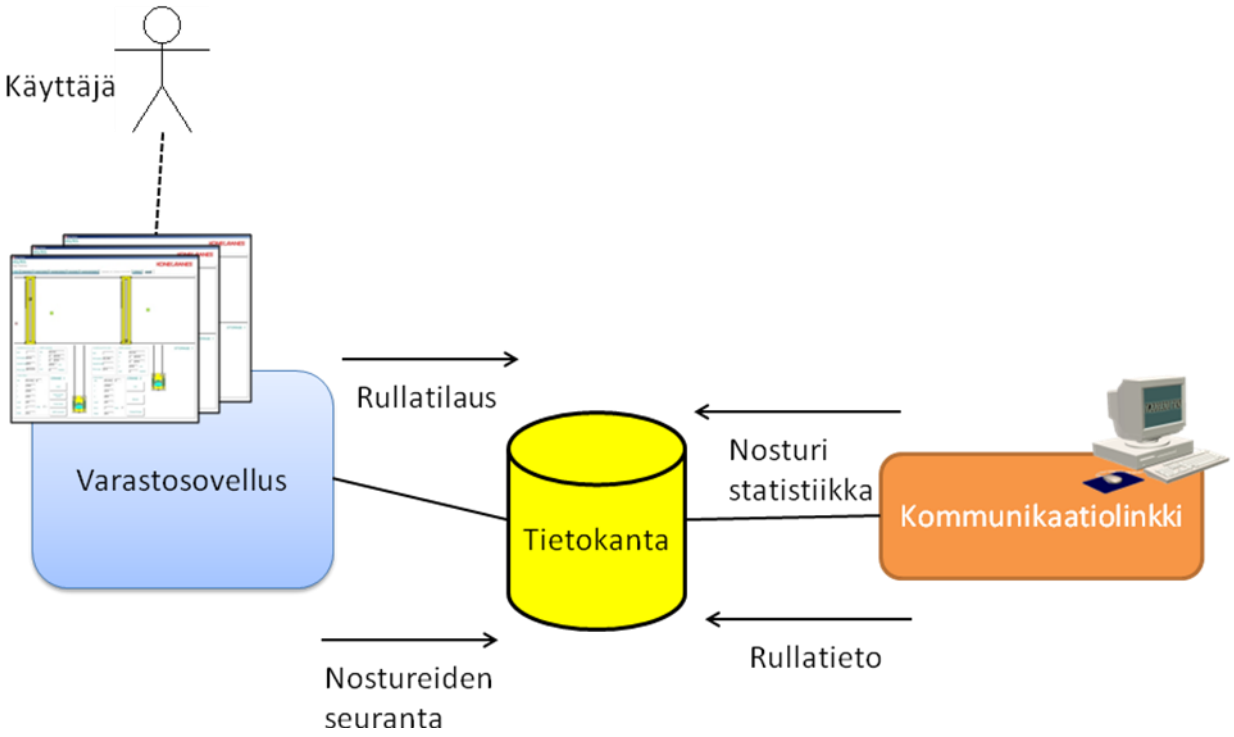


Kuva 8. Varastosovelluksen sovellusarkkitehtuuri

4.4 Varastosovelluksen toimintaperiaate

Varastosovelluksen avulla käyttäjä voi seurata pystyrullavaraston toimintaa, johon liittyy muun muassa nosturien- ja rullien seurantaa. Sovelluksen käyttöliittymä perustuu erilaisiin näkymiin, jotka kuvaavat varaston tilannetta esimerkiksi nosturien sijaintia, rullien sijainti- ja tilaustoimintaa sekä yksittäisen nosturin statistiikkaa.

Prosessikäyttäjällä voi olla tiettyjä oikeuksia eri toimintoihin, joita sovelluksen ylläpitäjä voi määrittää. Prosessikäyttäjiä voi olla useita, joten oikeudet riippuvat eri tilanteista ja käyttötarkoituksista (kuva 9).

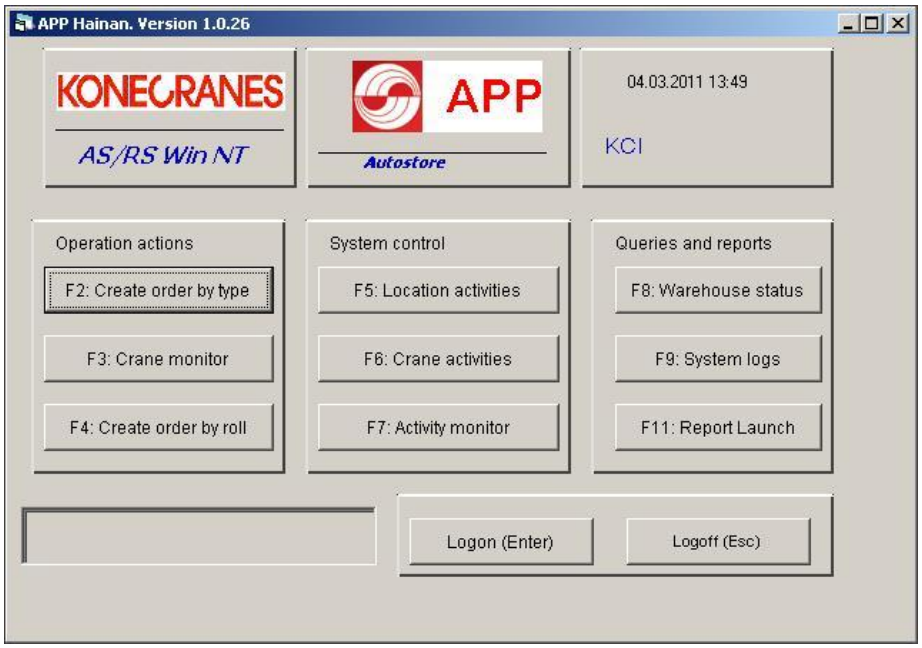


Kuva 9. Varastosovelluksen toimintaperiaate

4.5 Käyttöliittymä ja parannusehdotuksia (VB6)

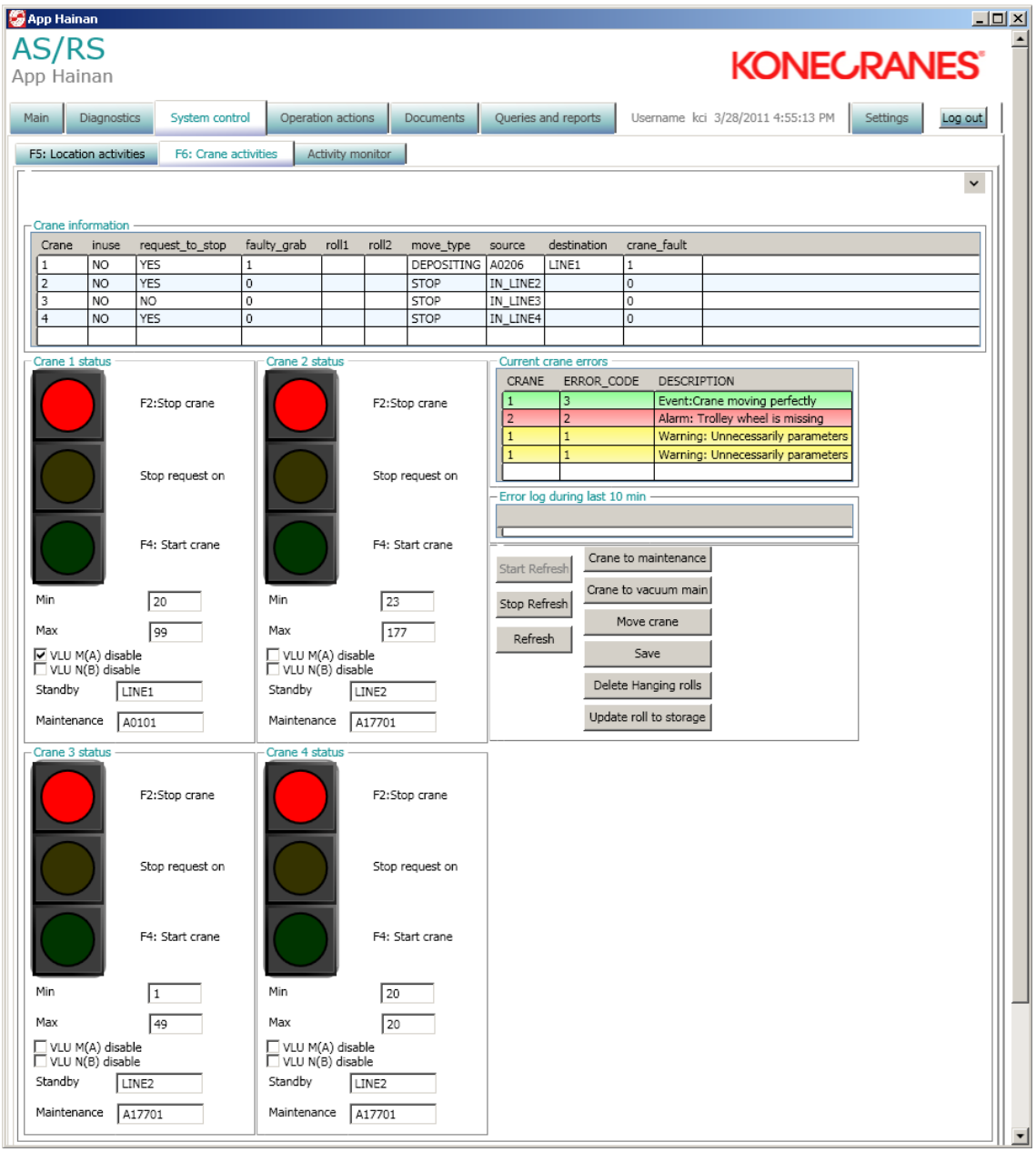
Päivitettävän varastosovelluksen käyttöliittymän ulkoasu on rakennettu MDI-tekniikalla, joka perustuu modaalisiin Windows-lomakkeisiin. Sovelluksen päänäkymä sisältää lomakkeen, joka koostuu muutamasta painikkeesta (Button). Painikkeet on jaoteltu sovelluksen eri toimintoihin.

Painikkeita painettaessa ruutuun ilmestyy modaalisia-ikkunoita, joista käyttäjä voi seurata pystyrullavaraston tilannetta muun muassa nosturin sijaintia, paperirullien statistiikkaa ja nosturin vikadiagnostiikkaa (kuva 10). Käyttäjä toimii erillisessä valvontahuoneessa, jossa varastosovellusta ylläpitävä tietokone sijaitsee.



Kuva 10. Pääsivun käyttöliittymä (Visual Basic 6)

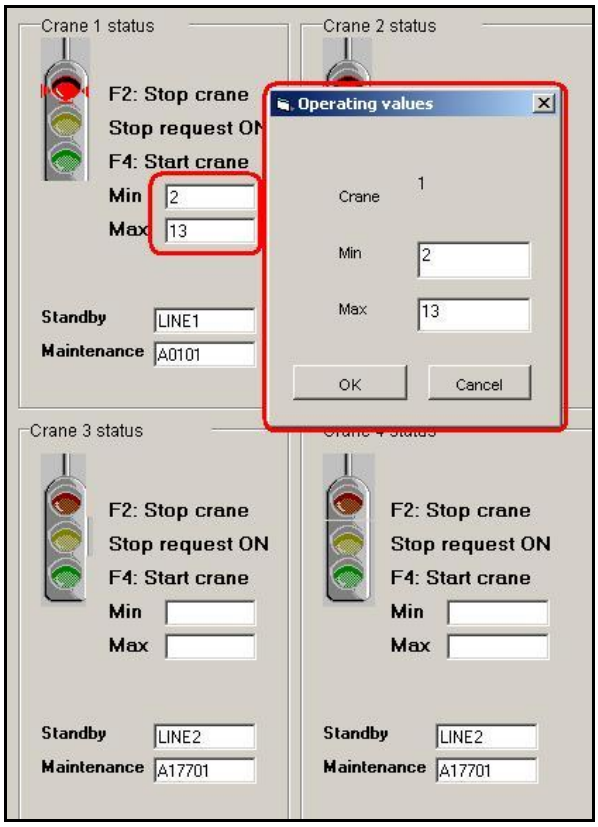
Käyttöliittymän ulkoasu toteutetaan VB.NET:ssä välilehtirakenteella, joka tuo siihen enemmän selkeyttä ja kontrollia (kuva 11).



Kuva 11. Välilehti-rakenne (.NET)

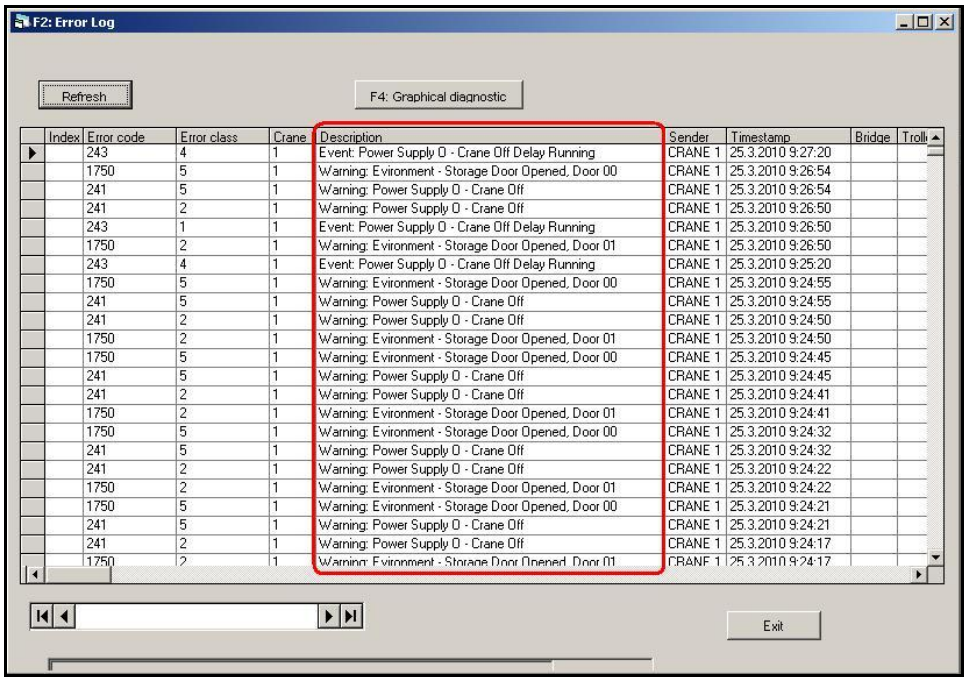
Crane activities -näkymällä voidaan seurata varastossa olevien nosturien toimintaa sekä siirtää nostureita annetuilla syötteillä. Nosturin toimintasädettä varastossa voidaan säätää Operating values **-**dialogilla, johon uudet parametrit syötetään. Dialogeilla toiminen voi kuitenkin osoittautua hankalaksi ja ohjelman käytettävyyttä olisi tällöin syytä parantaa. Sovelluksen VB.NET-versiossa osa dialogien toiminnasta on upotettu

näkymän käyttöliittymään sisälle esim. expander-komponentilla. Parannusehdotukset on merkitty punaisella (kuva 12).



Kuva 12. Nosturin aktiviteettinäkymä (Visual Basic 6)

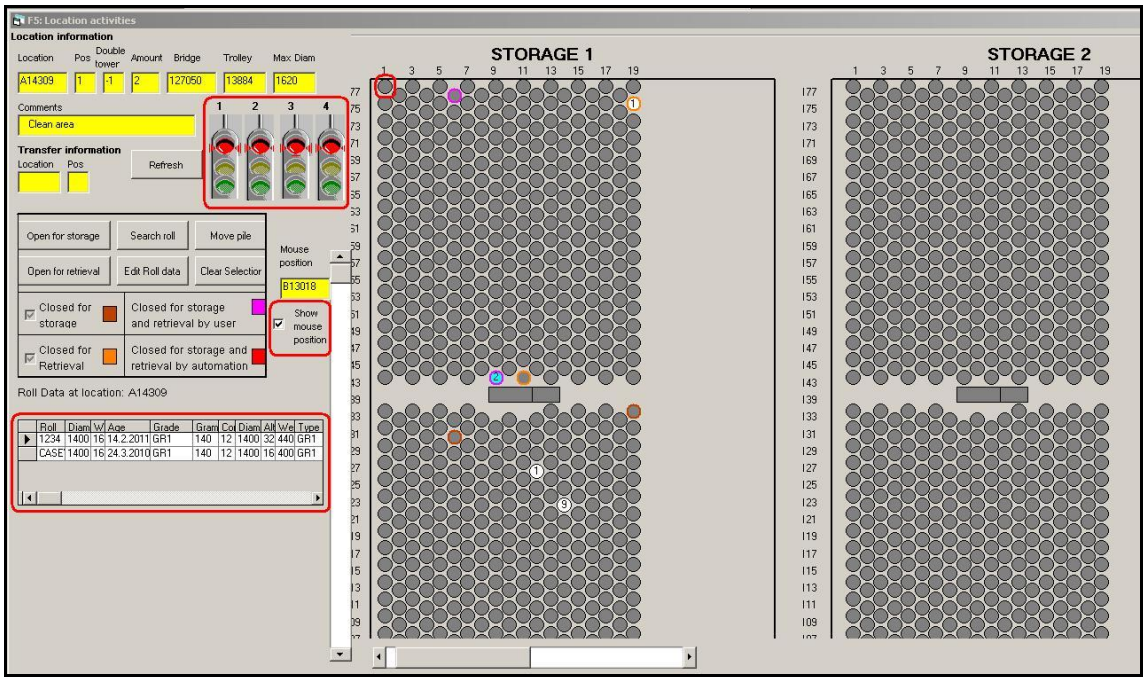
Nosturissa tapahtuvista vikailmoituksista vastaa vikadiagnostiikkanäkymä. Siitä voi seurata nosturissa tapahtuvien vikojen diagnostiikkaa. Viat jaetaan kolmeen eri asteeseen: Event, Warning sekä Alert. Vikailmoituksen tapahtuma aika voidaan määrittää käyttäjän antamien syötteiden perusteella. Tähän kuitenkin kuluu aikaa, sillä tässäkin näkymässä käytetään paljon dialogeja. Nämä voidaan korvata VB.NET:ssä toisenlaisilla menetelmillä kuten valmiiden kalenterien käytöllä esimerkiksi DatePicker-komponentilla (kuva 13). Vikojen luettavuutta voidaan myös parantaa. Tässä näkymässä vikojen asteita voitaisiin havainnollistaa paremmin erottelemalla ne väreiksi.



Kuva 13. Vikadiagnostiikkanäkymä (Visual Basic 6)

Paperirullien statistiikkaa voidaan seurata rullapaikkanäkymästä, jossa näytetään ylänäkymä paperirullavarastosta. Näkymään ladataan paperirullapaikat sekä paperirullien siirtelyyn vaadittavat kontrollit. Tässä näkymässä tarvitaan paljon uusia parannuksia. Paperirullapaikan havainnointiin näkymään käytetään ympyrän muotoista bittikarttakuvaa, jota monistetaan hunajakennomaiseen järjestykseen. Liikennevalot kuvaavat, onko nosturijärjestelmä käynnissä (vihreä valo palaa) vai pysähtyneenä (punainen valo palaa). Näkymän alaosassa on tietotaulu, joka näyttää valitun pinon paperirullatiedot. Nappuloiden vieressä on check-laatikko, jonka avulla paikkatietojen näkyvyyttä voidaan säätää (kuva 14).

Parannuksia .NET:iin on vektorigrafiikan hyödyntäminen liikennevaloihin sekä ympyräkuviointiin. Tietotaulun korvaaminen havainnollisemmalla tekniikalla esim. paperirullan sivuttaisnäkymä, johon rullan data upotettaisiin. Rullapaikkojen näkyvyyteen voidaan myös lisätä uusia toimintoja.



Kuva 14. Rullapaikkanäkymä (Visual Basic 6)

4.6 Tekninen arkkitehtuuri (.NET)

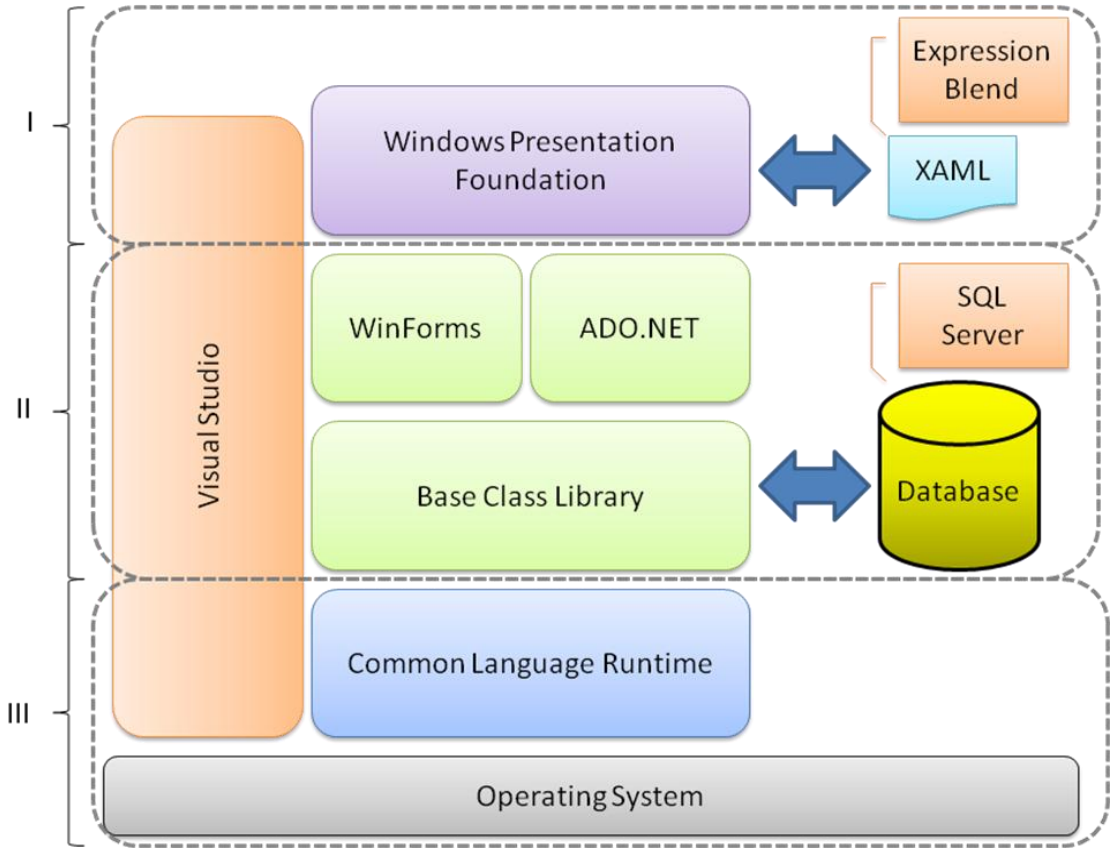
.NET Framework 3.5:n avulla kehitetyn varastosovelluksen tekninen arkkitehtuuri jakaantuu kolmeen eri tasoon. Arkkitehtuuri on ns. top-down tyyppinen rakenne, jossa korkeimmalla tasolla on sovelluksen käyttöliittymäkerros. Tasolla 1 on esitelty .NET Framework 3.5:en tarjoamat ohjelmointirajapinnat sovellukselle WPF, Visual Studio ja Expression Blend.

Varastosovelluksen käyttöliittymä toteutettiin WPF-ohjelmointirajapintaa hyödyntäen. Siinä ohjelmoitava logiikka on eroteltu selvästi käyttöliittymätasosta, mutta siihen voidaan toki vaikuttaa koodin puoleltakin (Code-behind). Tämä ilmenee .NET:ssä hyvän periytymistekniikan ansiosta. Käyttöliittymäkomponentit on jaoteltu selvästi eri nimiavaruuksiin eli ryhmittymiin.

Tasolla 2 on jaoteltu eri luokkakirjastot (Base Class Library), joita .NET Framework hyödyntää. Näitä ovat esim. Remoting, Threading, Reflection sekä I/O. Näiden avulla päästään helposti käsiksi erilaisiin järjestelmän toimintoihin. Muistia voidaan hallita I/O-kirjaston avullla. Threading-kirjastolla luodaan säikeitä käyttöjärjestelmän toimintoihin käytettäviin sovelluksiin. Remoting on kirjasto, joka tarjoaa toimintoja etäyhteyden

luomiseen. Reflection-kirjastolla voidaan luoda monen funktion joukko ja määritellä, mitä funktiota milloinkin käytetään. Taso on yhteydessä sovelluksen tietokantaan. Tietokantayhteys muodostetaan ADO.NET-rajapinnan tarjoamien komponenttien avulla.

Matalin taso 3 pitää sisällään CLR:n eli Common Language Runtimen, joka välittää ohjelmoidun koodin prosessorille luettavassa muodossa. Tämä taso pitää sisällään myös roskienkerääjän eli Garbage Collection. Sen tarkoitus on poistaa turhat ilmentymät (instance) muistista ja näin parantaa ohjelman ajonaikaista suorituskykyä (kuva 15). [8.]



Kuva 15. Tekninen arkkitehtuuri (.NET)

5 Visual Basic 6- ja VB.NET-ohjelmistomigraatio

Visual Basic 6- ja VB.NET - ohjelmistomigraatio voi olla pitkä prosessi yrityksessä, jossa .NET-kehittäjiä on vain muutama. Tämä riippuu kuitenkin paljolti käännettävän ohjelmiston laajuudesta ja käännettävistä moduuleista. On mahdollista, että joitakin moduuleja ei tarvitse välttämättä kääntää ollenkaan Visual Basic -kielen syntaksin muuttumattomuuden takia.

Ohjelmistomigraatio voi osoittautua hankalaksi, jos alkuperäisestä ohjelmasta ei ole olemassa minkäänlaista dokumentaatiota tai sovelluksen arkkitehtuuria vastaavia kuvia.

Tyypillinen ohjelmistomigraatio alkaa siten, että yritys haluaa itse tehdä oman sovelluksen käännöksen ilman .NET-kehittäjää käyttäen apuna Visual Studion tarjoamaa Migration Wizard-ohjelmaa. Tämän ohjelman avulla voidaan kääntää vanha VB6-sovellus uudeksi VB.NET-sovellukseksi. Tämä on kuitenkin huono lähestymistapa ottaen huomioon käännettävän ohjelmiston laajuuden ja monimuotoisuuden. Migration Wizardin avulla voidaan toki kääntää yksittäisiä ja koodiriveiltään lyhyitä luokkia, mutta ei ole suositeltavaa kääntää kokonaista ohjelmistoa. Tämän käännösohjelman käytöstä voi myös aiheutua koodiin lukuisia virheitä ja varoituksia joiden korjaamiseen kuluisi viikkoja tai pahimmassa tapauksessa jopa vuosia. VB6- ja VB.NET-tekniikan yhteensopivuus ei ole niin yksiselitteinen.

Paras tapa tehdä ohjelmistomigraatio on palkata tai kouluttaa yritykseen .NET-kehittäjä, joka voisi tutustua vanhaan järjestelmään ja sitä kautta luoda ohjelmistomigraatiolle suunnitelman käyttäen apuna vanhan ohjelmiston dokumentaatioita.

Ohjelmistomigraatio vaatii tarkan suunnitelman jokaiselle käännettävälle operaatiolle, luokalle sekä moduulille. Tämän lisäksi tehdään raportointia ohjelmistomigraation etenemisestä sekä myöhemmässä vaiheessa uuden sovelluksen testaamisesta. [12.]

5.1 Migraatiosuunnitelma

Migraatiosuunnitelma voidaan jakaa moneen eri vaiheeseen. Aluksi kääntäjän on hyvä tietää, mitä oikeastaan ohjelmasta pitäisi kääntää. Käännettävä VB6-sovellus voi pitää sisällään paljon eri tekniikoita esim. käyttöliittymäluokkia (UserControls), Web-ikkunoita (Web Classes) tai pelkästään ohjelman operaatioista ja logiikasta koostuvia tiedostoja. Tämän jaottelun jälkeen käyttäjän on helppo valita moduulinsa, jota lähteä kääntämään. Voi olla, ettei joitakin luokkia tarvitse kääntää ollenkaan.

Tyypillinen migraatiosuunnitelma pitää sisällään lisäämistä, poistamista, päivittämistä ja testaamista. Käytännössä ohjelmistomigraatio on sukua ohjelmistotuotannossa käytettäville ketterille menetelmille, jossa prosessi koostuu lyhyistä sykleistä.

Ohjelmistomigraatiossa käytetään 6 eri vaiheeseen jaoteltua migraatiosuunnitelmaa. Vaiheet on jaoteltu näin:

Vaihe 1. Päivitettävät moduulit ovat ne luokat ja moduulit, jotka kuuluvat ohjelmistomigraatioon. Joitakin luokkia tai moduuleja ei välttämättä tarvitse kääntää Visual Basic -kielen muuttumattomuuden takia.

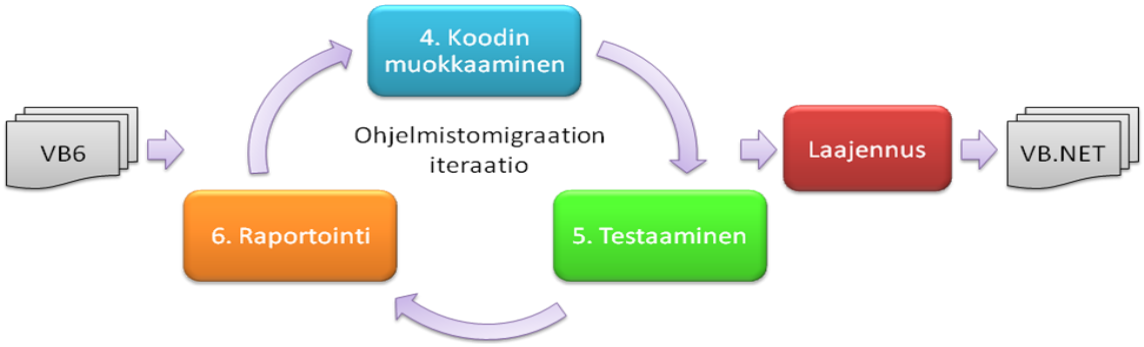
Vaihe 2. Luodaan täysin uusi tyhjä projekti Visual Studio -kehitysalustalla.

Vaihe 3. Käännetään vanha VB6-sovellus VB.NET-sovellukseksi Visual Studiossa Migration Wizard -käännöstyökalua apuna käyttäen.

Vaihe 4. Vanhan ja kopioidun koodin muokkaaminen. Tässä vaiheessa kopioidaan käännetty koodi uudelle VB.NET-projektille. Poistetaan turhat muuttujat. Tarkistetaan funktioiden tai operaatioiden laatu ja näkyvyys toisteisuuden välttämiseksi.

Vaihe 5. Pitää sisällään yksikkö ja moduulitestausta. Tehdään käännettyyn luokkaan testejä kopioimisen jälkeen. Testaaminen pitää sisällään funktioiden toimivuus- ja tehokkuustestausta, jatkuvaa yksikkötestaamista, jonka jälkeen projektin edistyessä myös moduulitestaamista ja luokkien välillä esiintyvän vuorovaikutuksen testaamista.

Vaihe 6. Raportoidaan mitä on tehty ja mitä on vielä tekemättä.



Kuva 16. Visual Basic 6- ja .NET ohjelmistomigraation iteraatio

Migraatiosuunnitelma on melko yksinkertainen kokonaisuudessaan. Ketteristä menetelmistä tuttu iteraatio syntyy vaiheiden 4, 5 ja 6 välille (kuva 16). Kun varastosovellus on käännetty VB.NET:iin, niin tämän jälkeen sitä voidaan halutessa laajentaa. Varastosovellukseen voidaan lisätä uusia ominaisuuksia .NET Frameworkin tarjoamilla komponenteilla. [13.]

5.2 Koodin validointi

Ohjelmistomigraation edetessä kääntäjän on hyvä kiinnittää huomiota tuottamaansa ohjelmiston laatuun VB.NET-ohjelmointikielellä. Tätä toimenpidettä kutsutaan koodin validoimiseksi, jossa tuotettua koodia varmennetaan ja todetaan hyväksi. Validoiminen tulee paremmin esille testausvaiheessa, jossa aluksi hyväksi todettua koodia vielä parannellaan. Oli kyseessä sitten järjestelmä-, moduuli- tai yksikkötestaus niin validoiminen on aina yhtä tärkeää.

Valmiiksi käännetyssä VB.NET-koodissa validointi perustuu eri sääntöihin. Jotkut validointisäännöt eivät vaikuta uuden sovelluksen toimivuuteen lainkaan ja näin ollen asia on enemmän kosmeettinen kuin rationaalinen.

Yksi validointisääntö on muuttujien nimeämiskäytäntö. Kun ohjelmistomigraatiota tehdään ja uuteen VB.NET-sovellukseen nimetään luokille muuttujia, niin nimeäminen noudattaa tiettyä sääntöä. Muuttujan nimen ensimmäinen kirjain täytyy olla kirjain, numero tai alaviiva tässä säännössä. Jos muuttujan nimi on todella pitkä ja sisältää

paljon sanoja, niin silloin tulisi erotella sanat isoilla alkukirjaimilla. Funktiot ja metodit on hyvä nimetä verbein, jotta ulkopuolinen tarkastelija saa pienen tiedon siitä, mitä toimenpidettä suoritetaan esim. CloseDialog. Rajapintojen nimet merkitään I-alkuisella kirjaimella ja nimet ovat adjektiivissa kuvaten, millainen rajapinta on esim. IPersistable.

Kokonainen funktio tai niiden joukko voidaan jäsentää siistiksi paketiksi region-validoinnin avulla. Region-validointi toimii siten, että liitetään yksi tai useampi sovelluksen toiminto määritellyn region-alueen sisään ja alue nimetään toimintoja kuvaamalla tavalla esim. #Region ”SaveFunctions”.

VB.NET-koodin kommentoinnissa käytetään yhtä heittomerkkiä (’). Tämän jälkeen koodirivi muuttuu vihreäksi eikä kääntäjä tulkitse täten riviä ohjelmakoodiksi. Kommentoiminen on myös osa käännösprosessia ja on hyvä merkitä näkyviin, mitä kukin toiminto suorittaa, jos tätä ei ole aikaisemmin tehty.

Koodin validoimiseen liittyy myös erikoismerkkejä kuten piste, kaksoispiste, huutomerkki ja et-merkki (&). Pisteellä kutsutaan muuttujan toimintoja ja ominaisuuksia. Kaksoispisteellä erotellaan lauseet toisistaan. Huutomerkkiä käytetään muuttujan oletusarvon määrittämiseen. Et-merkillä ketjutetaan muuttujia yhdeksi suurempiarvoisemmaksi muuttujaksi. [14.]

5.3 Testaus

Ohjelmistomigraation aikana on tärkeää tehdä yksikkötestausta käännettyihin luokkiin. Yksikkötestaus voi rajoittua vain luokan sisällä esiintyviin toimintoihin, mutta se on silti erittäin hyödyllistä ja antaa pohjaa myöhemmässä vaiheessa tulevaan moduuli- ja järjestelmätestaukseen.

Suurissa ohjelmistoprojekteissa on yleensä käytössä oma versionhallintansa, joten ohjelmistomigraatiossa uudelle VB.NET-projektille tulisi luoda versionhallinta. Versionhallintaa voi ryhmittää hyvin testauksen avulla. Kun testattava VB.NET-luokka on validoitu ja kaikki input- ja output-testit on suoritettu, niin lopuksi voidaan ohjelma ajaa .NET Frameworkin virtuaalikääntäjällä. Jos ohjelma on kääntynyt ongelmitta, niin projektin sen hetkinen revisio voidaan sitouttaa (Commit). Näin testausta ei välttämättä tarvitse

uusia samalle luokalle ja käännösprosessi pysyy hallinnassa paremmin, kun tiedetään testatut luokat ja moduulit.

Yksikkötestauksessa testataan käännetyn toiminnon sisältöä. Kyseessä voi olla joko funktio (Function) tai ali-funktio (Sub). Alifunktio eroaa funktiosta siten, että se ei palauta mitään arvoa takaisin kutsujalle. Toimintojen sisältä testataan silmukoiden toimivuutta sekä konstruktorin rakennetta.

Tämän testauksen avulla pyritään myös selvittämään, että toimiiko käännetyn moduulin tai luokan kaikki metodin kutsut samalla tavalla kuin vanhemmassa VB6-sovelluksessa. [3, s. 580 - 582.]

5.4 Ohjelmistomigraation ongelmat ja yhteensopivuusvirheet

Kun ohjelmistomigraatiota tehdään päivitysoperaationa Visual Studion tarjoamalla Migration Wizard -työkalulla, päivityksessä ei voida välttyä käännöksessä syntyviltä virheiltä ja ongelmilta. Käännöksessä asiat eivät välttämättä kuitenkaan tapahdu aina, niin kuin kääntäjä olisi alun perin suunnitellut. Kääntämisen yhteydessä syntyneet virheilmoitusten ja varoitusten määrä voidaan mitoittaa sovelluksen laajuuteen.

Yhteensopivuusvirheet on luokiteltu viiteen eri varoitus- ja virhetyyppiin. Jos virheet ovat vakavia, ne voivat estää päivityksen toteuttamisen.

Virhetyypit ja niiden selitykset:

1. Ominaisuus ei ole päivitettävää tyyppiä (virhe):

- VB.NET ei tue ominaisuutta. Tässä tapauksessa käännettävä koodi tulisi jättää nykyiseen Visual Basicin syntaksiin tai mahdollisesti päivittää ohjelmistoa perusteellisemmin.

2. Vaaditaan korjausta ennen päivittämistä (varoitus):

- VB.NET ei tue ominaisuutta. Nykyisen koodin voi kääntää, mutta se tulisi uudelleen korjata ennen päivitysoperaatiota VB.NET:iin.

3. Kehotetaan korjausta ennen päivittämistä (varoitus):

- Projekti tulisi tallentaa, jos koodia tai käyttöliittymäkomponentteja muokataan ennen päivitysoperaatiota.

4. Voidaan korjata ennen tai jälkeen päivityksen (varoitus):

- Koodia tulisi muokata. Tämän voi tehdä ennen tai jälkeen päivityksen.

5. Käännöstyötä vaaditaan päivittämisen jälkeen (informaatio):

- Koodia olisi syytä tarkastella tai muokata, mutta muutoksia ei voida toteuttaa ennen päivittämistä; esimerkiksi syntaksin korjaaminen tai olio-rakenteiden muokkaaminen.

Tyypillisiä VB.NET-yhteensopivuusongelmia:

1. Tiedon sitomisessa vaaditaan ADO-tietokantarajapintaa.

- Tiedonhallintakomponentit tukevat vain ADO-rajapintaa, vanhan RDO:n tai DAO:n sijasta. RDO- tai DAO-rakenteita tulisi muuttaa ADO-malliseksi rakenteeksi ennen päivittämistä.

2. Taulukko täytyy alkaa indeksistä 0 VB.NET:ssä

- VB.NET:n taulukon alaraja täytyy olla 0, muuten taulukot, joiden alaraja ei ole 0, muutetaan päivittämisen yhteydessä automaattisesti.

3. ByRef-viittausta parametreihin ei tueta VB.NET:ssa

- Väärä ByRef-viittaus voi johtaa arvaamattomaan funktion käyttäytymiseen. Kaikki parametreihin viittaavat määritykset tulisi toteuttaa ByVal-ominaisuutta käyttäen.

4. COM-metodi ei ole kutsuttavissa VB:NET:ssa.

- VB.NET tukee COM-luokan metodeja, mutta ei tue COM-moduulin metodeja. Tämän vuoksi VB.NET:iin on tehtävä manuaalisesti COM-metodeja vastaava rakenne.

5. Käyttöliittymäkomponenttia ei tueta VB.NET:ssa

- Käännettävää komponenttia ei ole valittavissa VB.NET:n omassa kirjastossa. Esimerkiksi VB6:en PictureBox-komponentti korvataan automaattisesti Image-komponentilla.

6. Poikkiviivoja ei tueta VB.NET:ssa

- VB.NET ei tue poikkiviivoja (Line Control). Päivittämisen jälkeen poikkiviivat on ylikirjoitettu OnPaint-metodilla.

7. Drag & Drop -toteutus täytyy uudelleen kirjoittaa VB.NET:ssa

- VB.NET tukee Drag & Drop -toteutusta, mutta oliomalli on hieman erilainen. Tämän vuoksi se täytyy kirjoittaa käsin uudelleen.

8. MDI-ikkunoiden yhteensopimattomuus VB.NET:ssa

- VB.NET ei tue MDI-ikkunoiden tapahtumia: MDIForm_click, MDI-Form_MouseDown, MDIForm_MouseMove ja MDIForm_MouseUp.

9. Moduulityyppiä ei tueta

- Moduulityyppiä ei tueta VB.NET:ssä. Esimerkiksi VB6:en DataReports. Tässä tapauksessa moduulityyppi tulisi säilyttää sellaisenaan.

10. Datatyyppiä ei tueta

- Currency- ja fixed-length -tyyppisiä datatyyppejä ei tueta. Esim. currency-tyyppinen muuttuja konvertoidaan automaattisesti decimal-tyyppiseksi muuttujaksi.

11. VB5-projektia ei voida päivittää

- Jos projekti on tehty VB6:sta aikaisemmalla VB5-tekniikalla, niin sitä ei voida päivittää. Se tulisi ensin päivittää VB6-projektiksi.

Käännösvaiheessa joutuu työskentelemään paljon erilaisten virhe- ja varoitusilmoitusten kanssa. Nämä ilmoitukset ovat kuitenkin osa ohjelmistomigraatiota ja antavat pientä suuntaa ja ohjeita kääntäjälle ohjelmistomigraation toteuttamiseen. [15.]

6 Visual Basic 6- ja .NET-järjestelmien eroavaisuudet

Graafinen .NET-käyttöliittymäkomponentti ei ulkoasultaan kovin paljoa eroa VB6:n komponentista. Eroavaisuuksia tulisikin näin tarkastella komponentin ohjelmoitavan koodirakenteen näkökulmasta.

Kielellisesti sovellukset eivät eroa juuri paljon, sillä Visual Basic -kielen syntaksi perustuu suurelta osin BASIC-ohjelmointikieleen. Suurin eroavaisuus järjestelmien välille syntyy .NET Frameworkin ja sen tuomien luokkakirjastojen myötä.

Ohjelmistomigraation jälkeen tehdyt laajennukset kasvattavat tekniikoiden eroavaisuuksia. Näin kasvaa myös .NET-pohjaisen varastosovelluksen kompleksisuus uusien tietorakenteiden ja komponenttien myötä.

Erovaisuuksia voidaan tarkastella monella eri tasolla. Alhaisimmalla tasolla tarkastellaan yksittäisten komponenttien eroavaisuuksia järjestelmien välillä. Keskitasolla kuvataan järjestelmien toimintojen kuten metodien ja funktioiden erovaisuuksia.

Ylimmällä tasolla kuvataan moduulien keskinäistä toimintaa sovelluksen välillä. Tähän tasoon voidaan luokitella myös käyttöliittymäkomponenttien graafinen eroavaisuus järjestelmien välillä.

6.1 Tekniset eroavaisuudet

.NET Framework:n tarjoama CLR (Common Language Runtime) -ajoympäristö on huomattavin ominaisuus .NET-pohjaisten sovellusten kehittämisessä. CLR:ään kuuluu myös hyväksi todettu roskienkeruutoiminnallisuus, jonka avulla ohjelmoijan ei tarvitse huolehtia jokaisen luodun olion tuhoamisesta itse, vaan ohjelma tekee sen automaattisesti. Näin ollen VB.NET on myös enemmän oliopohjainen kieli kuin VB6. [7.]

Ajonaikaiset virheiden käsittelyyn liittyvät ominaisuudet VB.NET:ssä eivät ole niinkään kehittyneet, mutta syntaksi on helpommin tulkittavissa ja eroavaisuudet ovat huomattavat. Esim. VB6:ssa käytetään ’On Error Goto’ -syntaksia, kun taas VB.NET:ssa syntaksi on ’Try…Catch…Finally’. [3, s. 20 - 21.]

Yksi merkittävä muutos VB.NET:ssä on muutamien tietotyyppien uusi rakenne. Esim. Integer-tietotyypin pituus on kaksinkertaistunut 16 bitistä 32 bittiin. Long-tietotyyppi on myös kaksinkertaisunut 32 bitistä 64 bittiin. 16-bittinen kokonaisluku tunnetaan nykyään VB.NET:ssa tietotyyppinä ”Short”. [8.]

Tietokantarajapintana VB6:ssa käytetään ADODB-rajapintaa, jonka avulla voidaan luoda tietokantaa käyttäviä sovelluksia. VB.NET:ssä rajapintana voidaan hyödyntää ADO.NET -tietokantarajapintaa. Rajapinnat eivät eroa paljon tekniikoiden välillä, mutta toiminnallisuus hieman. [16.]

.NET Framework tarjoaa kieliriippumattoman ajoympäristön. Eli VB.NET-projektissa voidaan käyttää VB.NET-kielen lisäksi myös muita .NET-perheeseen kuuluvia ohjelmointikieliä kuten C#:aa, C++:aa ja Visual C:tä. Kaikki sovelluksessa käytettävät kielet käännetään IL-kieleksi, mutta ne eivät käänny automaattisesti. (Intermediate Language). [8.]

VB6 ei tue säikeitä sovelluksissa. VB.NET:ssa voidaan taas luoda monisäikeisiä sovelluksia. VB6:lla voidaan luoda vain työpöytäsovelluksia (Desktop Windows Application), kun taas NET Framework tarjoaa rajapintoja web-pohjaisille ja yleisille Windows-sovelluksille. [8.]

6.2 ActiveX kontrollit .NETssä

ActiveX on ohjelmakoodia sisältävä palanen, joka muodostuu useammasta objektista käyttämällä ActiveX-teknologiaa. Ohjelma voi koostua jo olemassa olevista komponenteista, jotka on luotu Microsoft Office -ohjelmissa, koodikomponenteissa, ActiveX-asiakirjoissa tai ActiveX-objekteissa (OLE-objekti). Ohjelmoija voi halutessaan tehdä myös oman ActiveX-komponentin, mikäli kehitysalusta tarjoaa tähän työkalun. [2, s. 499.]

.NET Framework tukee ActiveX-komponentteja, mutta ei suoranaisesti. .NET Framworkin upgrade wizard -käännöstyökalulla voidaan kääntää VB6-lomakkeen (Form) ActiveX-komponentit .NET lomakkeeseen sopivaksi. Käännöksen yhteydessä uuden .NET:n ActiveX -komponentin nimeäminen on kuitenkin tehtävä manuaalisesti. .NET Frameworkin periytymistekniikan vuoksi joidenkin ActiveX-komponenttien ominaisuudet

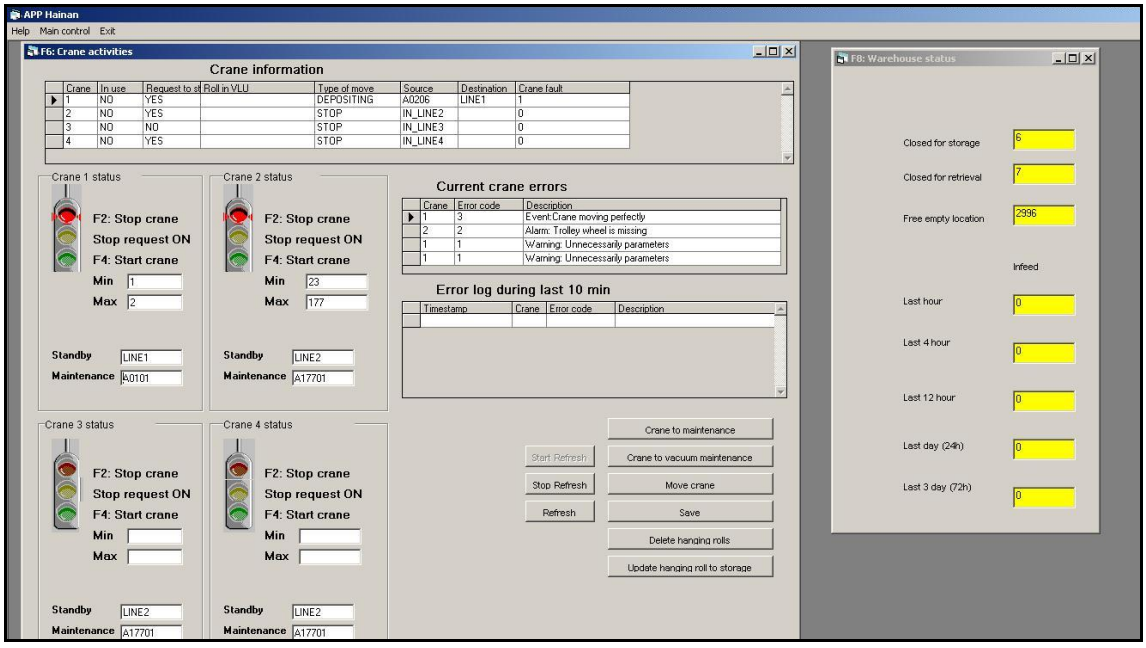
ovat eri nimisiä kuin VB6:ssa. Komponenttien toiminnallisuudet ovat kuitenkin samat järjestelmien välillä. [18, s. 19 - 20.]

6.3 Visual Basic 6 -pohjaisen sovelluksen MDI-rakenne

VB6-ohjelmointi perustuu suurelta osin lomakkeisiin ja se on suunniteltu RAD-mallin mukaiseen (Rapid Application Development) ohjelmistokehitykseen. Puhutaan ns. nopean ohjelmoinnin kehityksestä, jossa ohjelma luodaan mahdollisimman nopeasti säästäen aikaa käyttöliittymäkomponenttien koodaamiselta.

Visual Basic -kehitysalustalla voidaan sovellukseen lisätä lomakkeita hiiren napsautuksella. Liiallisen klikkaamisen vuoksi lomakkeita voi tulla liikaa sovellukseen ja käyttöliittymästä erittäin sekava. Toisaalta monen lomakkeen käyttöliittymä tarjoaa helpon tavan seurata ohjelman toimintoja rinnakkain.

Jos ohjelman käyttöliittymä perustuu vain yhteen lomakkeeseen, niin puhutaan ns. SDI-mallista (Single Document Interface). Toisaalta yhden Windows-ikkunan sisällä voi olla monta lomaketta jolloin puhutaan ns. MDI-mallista (Multi Document Interface) (kuva 17). [2, s. 17.]

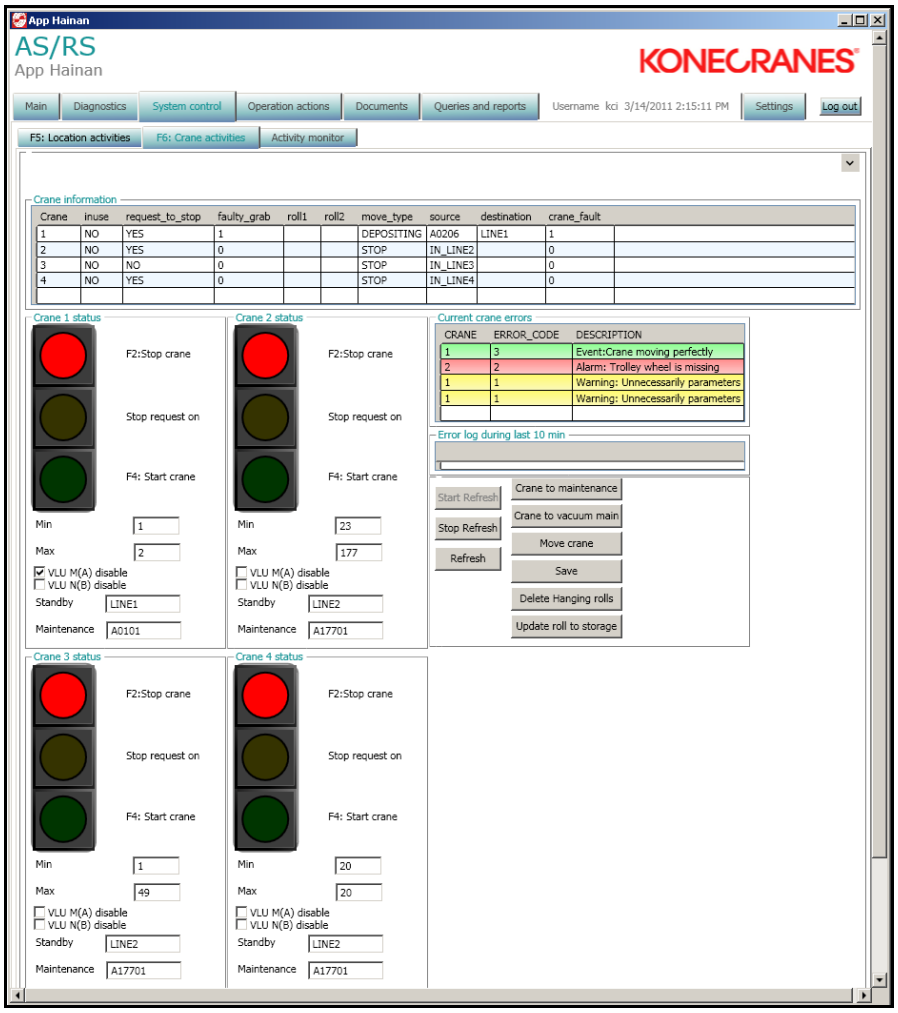


Kuva 17. Käyttöliittymän MDI-malli (VB6)

6.4 Visual Basic .NET -pohjaisen sovelluksen välilehtirakenne

Jos vanhan VB6-pohjaisen sovelluksen käyttöliittymärakenne perustuu MDI-mallin mukaiseen rakenteeseen, niin se on hyvä korvata VB.NET-pohjaiseen sovellukseen välilehtirakenteella (TabItems). Näin käyttöliittymästä saadaan paljon selkeämpi ja helpompi käyttää. Hiirellä tehtyjen klikkausten määrä vähenee huomattavasti, kun ei tarvitse sulkea ja avata samoja ikkunoita uudelleen ja uudelleen. Ikkunoiden samanaikainen seuraaminen ei ole tällöin mahdollista, mikä onkin välilehtirakenteen haittapuoli. Tämän ongelman voisi korjata erilaisilla erotinkomponenteilla (Separator), mutta ohjelman käytettävyys saattaa tällöin kärsiä.

.NET-pohjaisessa sovelluksessa välilehtiratkaisu voi olla myös hyvä ohjelman värimaailman kanssa, sillä välilehdelle voidaan määrittää omat tyyli-ominaisuudet (Style). Välilehtikontrollilla on ns. riippuvuusominaisuudet (Dependency properties). Tämän ominaisuuden ansiosta välilehden sisällä olevat lapsielementit (Child) ovat riippuvaisia periytyvästä vanhemmasta elementistä (Parent). Välilehden tyyliominaisuuksia voidaan siis periyttää sen alla oleville lapsielementeille (kuva 18). [19.]



Kuva 18. Käyttöliittymän välilehtirakenne (VB.NET)

6.5 Korvaavat komponentit

Kaikkia VB6-pohjaisessa sovelluksessa esiintyviä komponentteja ei voida suoraan korvata VB.NET-pohjaiseen sovellukseen siirryttäessä. Näihin tapauksiin löytyy kuitenkin usein hyvin yksinkertainen ratkaisu. Komponentit korvataan aivan uusilla .NET-komponenteilla, joissa niiden tietorakenne poikkeaa aiemmista VB6-sovelluksen käyttämistä komponenteista. Komponentti voi olla myös hyvin samanlainen, jolloin sitä ei välttämättä tarvitse korvata. [20.]

Korvattavat komponentit voidaan jakaa eri ryhmiin. Näitä ryhmiä ovat:

- käyttöliittymäkomponentit
- tietokanta -ja logiikkakomponentit.

6.5.1 Käyttöliittymäkomponentit

VB6:n ja VB.NET:n käyttöliittymäkomponenttien eroavaisuudet voivat olla hyvinkin suuria ja korvaavia komponentteja tekniikoiden välillä on paljon. Joidenkin .NET-komponenttien perustoiminnallisuudet saattavat erota VB6:en komponenteista niiden käytettävyyden takia. Esimerkiksi VB6:n Shape-piirtokomponentti voidaan korvata VB.NET:ssa lukusilla vastaavilla piirtokomponenteilla, jotka kuuluvat Shapes-nimiavaruuteen. Seuraavassa taulukossa on esitelty muutamia korvaavia komponetteja VB6:n ja VB.NET:n välillä. [21.]

Taulukko 2. Käyttöliittymäkomponentit

VB6	**.NET (WPF)**
TextBox	System.Windows.Controls.TextBox, System.Windows.Controls.PasswordBox
OptionButton	System.Windows.Controls.RadioButton
Frame	System.Windows.Controls.GroupBox
CommandButton	System.Windows.Controls.Button
Shape	System.Windows.Shapes.Line System.Windows.Shapes.Ellipse System.Windows.Shapes.Rectangle System.Windows.Shapes.Path
PictureBox	System.Windows.Controls.Image

VB.NET-pohjaisen sovelluksen käyttöliittymään voidaan lisätä myös lomakekomponentteja (Form). [21.]

Taulukko 3. Lomake (Form) -komponentit

VB6	**.NET (Forms)**
ImageCombo	ComboBox
Frame	GroupBox, Panel
CommandButton	Button
CommonDialog	FileDialog, PrintDialog,

Menu	MainMenu, ContextMenu
MonthView	MonthCalendar
Slider	TrackBar
UpDown	NumericUpDown, DomainUpDown

6.5.2 Tietokanta- ja logiikkakomponentit

Tietokantayhteyteen löytyy myös muutamia ratkaisuja VB6:en ja VB.NET:n ohjelmistomigraatiossa. Tietokantayhteyteen on viisainta käyttää ADO.Connection-rajapintaa sen hyvän yhteensopivuuden vuoksi. Vanhan VB6-pohjaisen sovelluksen DAO.Connection-komponentit on paras korvata ADO-tietokantakomponenteilla. Jotta kääntäminen sujuisi nopeasti ja ongelmitta, niin on hyvä tietää, mitkä vanhat DAO parametrit ja tietokantakutsujen rakenteet vastaavat ADO:n rakennetta.

.NET Framework 3.5:n mukana ilmestyi LINQ-ohjelmistokomponentti, joka mahdollistaa datakyselyt .NET-ohjelmointikielessä. Sen avulla voidaan tehdä kyselyjä muihinkin metadataa käyttäviin tietorakenteisiin kuin tietokantaan: esimerkiksi XML- ja taulukkodokumentteihin. VB6-pohjaisen sovelluksen recordsource-tietokantaliitännän voi korvata LINQ:n komponentilla DataContext, jonka sisältä metadatan tietueet löytyy. [22.]

Taulukko 4. Tietokanta- ja logiikkakomponentit

VB6	**.NET (.NET Framework 3.5)**
DAO.Connection	ADODB.Connection
Recordset	ADODB.Recordset
QueryDef	ADODB.Command
SQL	CommandText
Timer	DispatcherTimer
Adodb.RecordSource	System.Data.Linq.DataContext

6.5.3 Tietotyypit

Oletuksena VB6:n muuttujat ovat variant-tyyppisiä. Variant-tyyppinen tieto voi olla numeerinen, päivämäärä tai merkkijonotyyppinen. Kun muuttujan tyypitys on suoritettu, niin sen jälkeen tiedetään, minkälaista tietoa muuttujaan tulisi tallentaa. Seuraavassa taulukossa on esitelty muutamia VB6-tietotyyppejä ja korvaavat .NET-tietotyypit. [23.]

Taulukko 5. Tietotyypit

VB6	**.NET**	**Varastointijako (tavu)**	**Arvoalue**
Integer	Int32	4	-2,147,483,648-2,147,483,647
Long (long integer)	Int64	8	-9,223,372,036,854,775,808-9,223,372,036,854,775,807 (9.2...E+18 †)
Short (short integer)	Int16	2	-32,768 - 32,767
UShort	UInt16	2	0 - 65,535

7 .NET-tekniikan tuomat parannukset

Jos lähtisi luettelemaan kaikkia .NET-tekniikan tuomia parannuksia ohjelmistokehitykseen, niin niistä saisi koottua oman kirjan. Listaan tässä vain muutamia tärkeimpiä parannuksia, jotka ovat auttaneet .NET-sovelluksen kehittämisessä.

Huomattavin parannus on .NET Framework, jonka eri palikoista VB.NET-projekti koostuu. Sen tärkein komponentti on CLR-ajoympäristö, joka huolehtii muistinhallinnasta ja roskienkeruusta. Tämän lisäksi se pitää ohjelman stabiilina ja turvallisena.

VB.NET on enemmän olio-pohjainen ohjelmointikieli kuin edeltäjänsä VB6. Se näkyy kontrollien laajana periytymishierarkiana .NET Frameworkin tuoman luokkakirjaston myötä.

ADO.NET:n tietokanta-rajapinta antaa myös huomattavan lisäarvon VB.NET-pohjaiselle sovellukselle. Se mahdollistaa XML-dokumenttien yhteensopivuuden ohjelman ja tietokannan välisessä tiedonkäsittelyssä.

.NET tukee monisäikeistä tekniikkaa (MultiThread) VB6:n yksisäikeisen sijasta. Tämä mahdollistaa monen eri toiminnon samanaikaisen suorittamisen parantaen sovelluksen käytettävyyttä ja nopeutta.

Turvallisuusasioissa .NET on hieman edellä Visual Basic 6 -tekniikkaa. .NET Framework tarjoaa ohjelmistojen turvallisuuteen tarkoitetun uuden turvallisuusmallin (Code Access Security). Tämä toimii siten, että tietyt ohjelmistossa olevat koodialueet ovat turvattu ja niille on asetettu tietyt oikeudet. Esimerkiksi käyttäjä voi asettaa komponenteille tiettyjä oikeuksia tietokantaan pääsyä varten. [26.]

7.1 Käytettävyys

VB.NET-pohjaisen sovelluksen toteutukseen käytettiin .NET Frameworkin WPF-ohjelmointirajapintaa. WPF-projektissa painopiste on enemmän sovelluksen käyttöliittymän suunnittelussa, joka on toteutettu XAML-merkintäkielellä. Käyttöliittymäkomponenttien muuntautuvuus on laaja ja niissä käytetyt graafiset ominaisuudet skaalautuvat tarkasti eri näytöille.

Käytettävyyttä VB.NET-pohjaisessa sovelluksessa voidaan lisätä myös kopioimalla reaalimaailmasta vaikutteita kustomoitujen kontrollien ansiosta. Niistä voidaan rakentaa reaalimaailmaa jäljittteleviä kokonaisuuksia, niin toiminnallisuudeltaan kuin ulkoasultaan. Tämä on yksi käyttöliittymäsuunnittelun peruslähtökohdista. Esimerkiksi laskin toimii parhaiten, jos sen osat muistuttavat reaalimaailman osia.

Hyvän käytettävyyden takaa myös se, että asioita voidaan näyttää selittämisen sijaan. WPF-rajapintaa käytettäessä tämä onnistuu parhaiten animaatioiden ja siirtymien avulla. Siirtymien avulla voidaan osoittaa ohjelmassa syntyneet virheet. Nämä voivat olla virhettä kuvaavia graafisia kuvia tai ikoneita. Tällä tavoin voidaan välttää selityksiä ohjelmassa, jotka saattavat käyttäjältä jäädä huomaamatta.

Visual Studion IntelliSense-toiminto tarjoaa .NET-kehittäjälle nopean tavan kirjoittaa ohjelmointikoodia. Se antaa välittömästi palautetta jos koodiin on syntynyt virheitä. Sen avulla voidaan myös täydentää koodia, jos käyttäjä on antanut vain osan koodirakenteesta.

Käytettävyyttä voidaan lisätä .NET-kontrollien lukuisten tapahtumankäsittelijöiden avulla. Esimerkiksi Button-kontrollin click-tapahtuma voidaan aktivoida tavallisen hiiren näppäimen lisäksi jopa ääniohjauksella. [24; 25.]

7.2 XAML-merkintäkieli ja kontrollien periytyminen

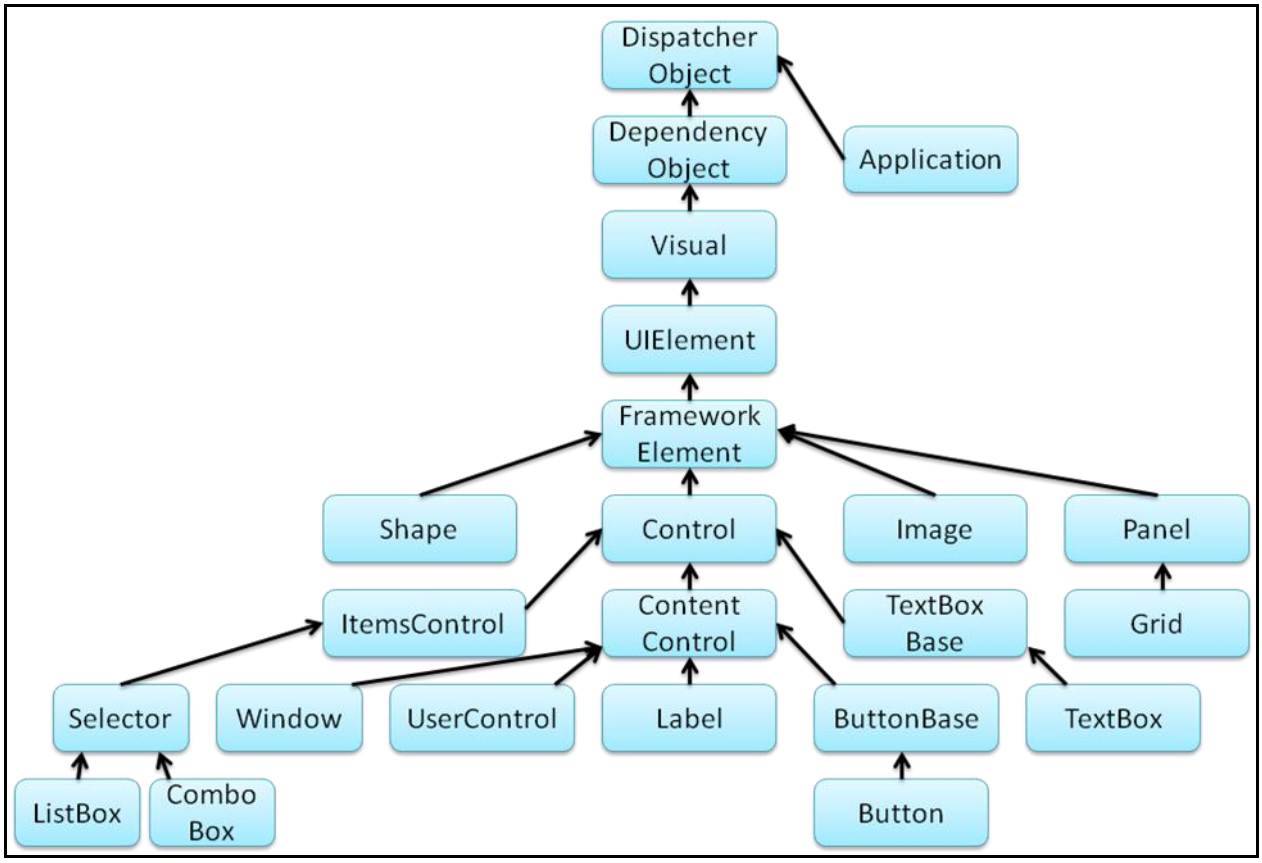
XAML (eXtensible markup language) on deklaratiivinen merkintäkieli, jolla luodaan WPF-rajapintaa käyttävän VB.NET-projektin käyttöliittymä. Sen avulla voidaan luoda näkyviä käyttöliittymäkomponentteja, jotka ovat erillään ohjelman logiikasta, joka on piilotettu käyttöliittymän taakse (Code behind). WPF-projekti voidaan luoda Visual Basic- tai C#-ohjelmointikielellä.

XAML-komponenteista koostuva käyttöliittymä on eräänlainen osaluokka (Partial class), jota kääntäjä kutsuu, kun komponentteja tulostetaan näytölle. Olio-ohjelmoinnille tuttua ilmentymien luontia XAML:ssa ei tarvitse tehdä ollenkaan, vaan XAML edustaa suoraan objektien ilmentymiä käytettävässä ohjelmassa. Tämä poikkeaa paljon muista tavanomaisista merkintäkielistä, jotka eivät yleensä ole suoraan tyypitettyjä sovellukseen.

XAML:n erikoisuutena voidaan pitää sen jaottelua eri osa-alueiden välille ohjelmistotuotannossa. Tekniikan mukana on ilmestynyt markkinoille lukuisia graafisia työkaluja, joiden avulla voidaan luoda graafisia käyttöliittymiä (GUI) ottamatta kantaa ohjelman toteutuksessa esiintyvään logiikkaan ja koodiin. Käytetyin kehitysalusta XAML-merkintäkielen tekoon on Microsoftin kehittämä Visual Studio ja .NET Framework 3.5 -komponenttikirjasto. Esimerkiksi yrityksessä, jossa on ohjelmistosuunnittelijoita sekä graafisia suunnittelijoita, ohjelmistokehitykseen tarkoitetut tehtävät voitaisiin jakaa näiden kesken. Yksi ohjelmistoprojekti voitaisiin hajottaa pienemmiksi osakokonaisuuksiksi eri toimijoiden välillä. VB.NET- ja WPF-yhteensopivuuden vuoksi tämä nopeuttaa logiikan ja käyttöliittymän integraatiota.

XAML-kieli muistuttaa html-kieltä sen syntaksin takia. Komponentit alkavat ja loppuvat aina tageihin (Tags). Komponenttien yhteinen rakenne koostuu hierarkkisesta pyramiidistä jossa päällä on aikuiselementtejä (Parent) ja niiden alla lapsielementtejä (Child). XAML on merkkiriippuvainen kieli (case sensitive) ja kaikki kontrollit ovat yhteydessä suoraan .NET Frameworkin kirjaston System.Controls luokkaan, joka tarjoaa rajapinnan XAML-komponenteille. Komponenteilla on käsittelijät (Handlers) ja tapahtumat (Events), jotka ovat komponentin jäsenfunktioita.

XAML on täysin olio-pohjainen kieli, joten sen komponentit voivat periä ominaisuuksia muilta komponenteilta. Esimerkiksi BCL:n kirjaston System.Windows.Controls.Button-luokka periytyy System.Windows.Controls.ButtonBase-luokasta ja tämä taas periytyy luokasta System.Windows.Controls.ContentControl. ControntControl periytyy FrameworkElement-luokasta ja tämä taas periytyy UIElement-luokasta. Alimman tason kontrolleilla on vähän omia ominaisuuksia, mutta sitäkin enemmän ylemmän tason ominaisuuksia periytymisestä johtuen (kuva 19). [20; 27; 28, s. 3 - 4.]



Kuva 19. WPF-kontrollien luokkahierarkia

7.3 Aritmeettisten operaatioiden oikopolut

VB.NET-ohjelmointikieleen on tullut muutamia lisäominaisuuksia .NET Frameworkin myötä. Yksi näistä ominaisuuksista on aritmeettisten operaatioiden oikopolut, joita Visual Basic 6:ssa ole. Näitä oikopolkuja on hyödynnetty ohjelmistomigraatiossa ja näin hyödynnetty VB.NET-pohjaisen sovelluksen suunnittelussa.

Oikopolkujen käyttö ohjelmakoodissa:

Visual Basic 6 syntaksi:

```
Dim myString As String
myString = myString & "SomeText"
```

Saman asian voi näin kirjoittaa hieman lyhyemmin VB.NET:ssa hyödyntäen oikoteitä aritmeettisissa operaatioissa.

VB.NET syntaksi:

```
Dim myString As String
myString &= "SomeText"
```

VB.NET:n aritmeettisten operaatioiden oikopolut ovat &=, *=, +=, -=, /=, \= ja ^=. [18, s. 12.]

7.4 Monisäietekniikka

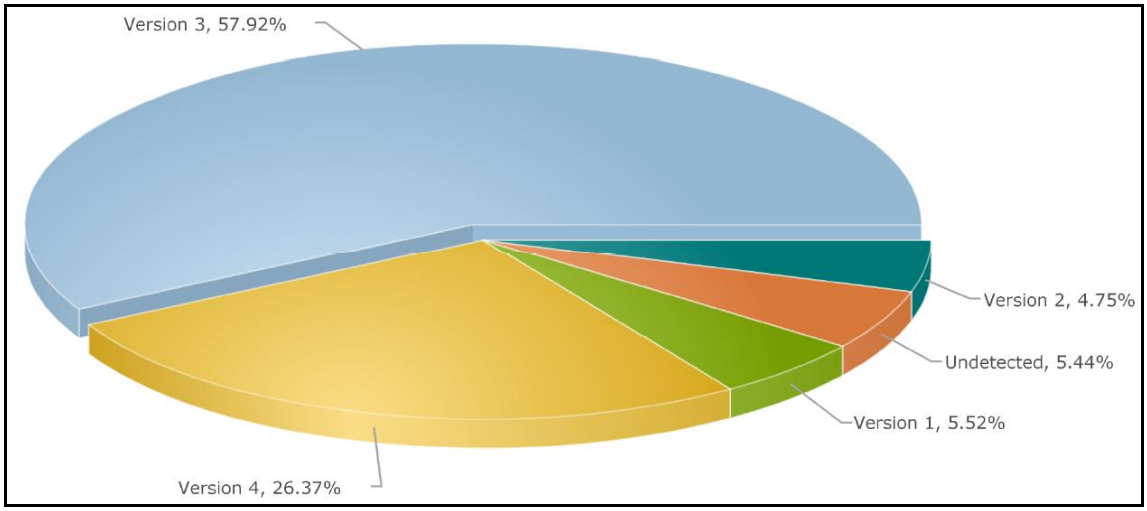
Monisäietekniikka tarkoittaa usean prosessin samanaikaista suorittamista sovelluksessa. Tekniikka on yksi tärkeimmistä .NET:n ominaisuuksista ja sopii käyttöliittymäsuunnitteluun lisäten enemmän herkkyyttä lomakkeiden selaamiselle. Sen avulla voidaan priorisoida operaatioita ja näin saadaan optimoitua paras mahdollinen suoritusaika. Sen käyttö on erittäin suotavaa nykyisissä tuplaydin prosessoreissa. [28.]

7.5 Asiakastuki

Microsoft tarjoaa täyden tuen .NET Frameworkille ja kaikille sen alla pyöriville tekniikoille kuten esimerkiksi VB.NET:lle, Visual C#:lle ja Visual C++:lle. Tämä on myös yksi

hyvä syy päivittää vanha VB6-sovellus uuteen, sillä Microsoft ilmoitti VB6-tuen loppuvan 8.4.2008 ja samoin sen ohjelmistokehitysalustan tuki loppui.

Oheisessa kuvassa (kuva 20) on piirakkadiagrammi koko .NET Framework -tuesta. Tuen määrä jakautuu versioittain .NET Framework 1, 2, 3 jne. Tutkimus on tehty ajalle 1.6.2010 - 10.11.2010 ja tutkimus rajoittuu vain Internet Explorer -käyttäjien tekemiin tukipyyntöihin. [30.]

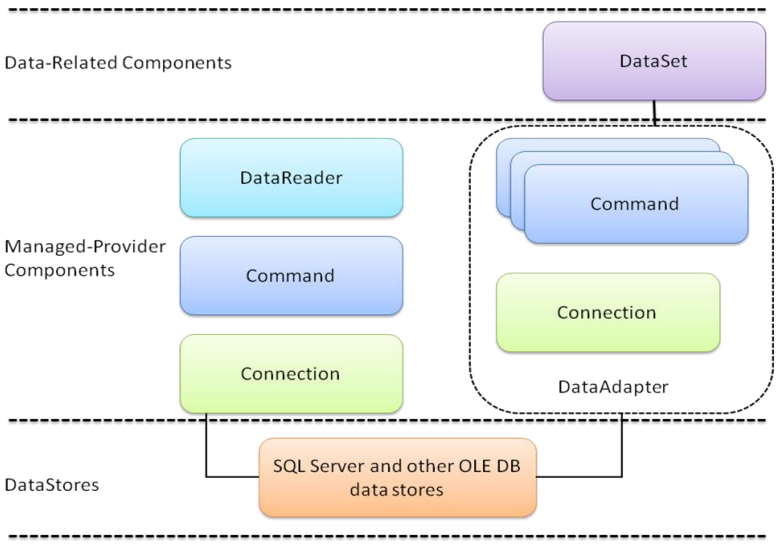


Kuva 20. .NET Frameworkin tuki versioittain

7.6 ADO.NET-tietokantarajapinta

ADO.NET-tietokantarajapinta on yksi .NET Frameworkin kirjastoon kuuluva kokoelma erilaisia tietokantayhteyteen tarkoitettuja ohjelmistokomponentteja. Sitä käytetään tiedon muokkaus-, tallennus- ja päivitystoimintoihin relaatiotietokantajärjestelmissä.

Sen erikoisuus piilee siinä, että sen avulla pääsee yhteydettömään tietokantaan myös käsiksi DataSet-olion avulla. Tietokannasta haetut tiedot tallentuvat automaattisesti DataSet-olioon ja tietoentiteetti on yksi olion ominaisuuksista. [31.]



Kuva 21. ADO.NET-arkkitehtuuri

Arkkitehtuuri jakautuu kolmeen tasoon (kuva 21). Ylin toimii client-tasolla, jossa on sisältökomponentti DataSet. Tämä pitää sisällään relaatiokannan tietoja, jotka näkyvät käyttäjälle, joita käyttäjä voi halutessaan muokata, päivittää tai poistaa. DataSet on siis yhteydetön objekti joka saa tietonsa DataAdapter-olion kautta. DataAdapter koostuu ominaisuuksista, jotka pitävät sisällään tietokantakomentoihin liittyviä funktioita.

Komponentteja hyödynnetään VB.NET-pohjaisessa sovelluksessa tietokannan ja bisneslogiikan välillä. Bisneslogiikka sitoo tiedon ja siirtää sen eteenpäin käyttöliittymätasolle käyttäjän tarkasteltavaksi. [32; 33 s. 20 - 22.]

7.7 WPF-ohjelmointirajapinta

WPF (Windows Presentation Foundation) on graafinen rajapinta, jota voidaan myös nimittää eräänlaiseksi alijärjestelmäksi. Sen avulla mallinnetaan käyttöliittymäkomponentteja Windows-pohjaisissa sovelluksissa.

WPF rakennettiin alun perin osaksi .NET 3.0-ohjelmistokomponenttikirjastoa ja on edelleen .NET:n uusimmissa versioissa mukana. WPF on rakennettu DirectX-grafiikkamoottorin päälle, joka tarjoaa käyttöliittymissä käytettäviä ominaisuuksia kuten läpinäkyvyys, häivytykset ja muunnokset. Nämä ominaisuudet tuovat modernille .NET-projektille enemmän lisäarvoa ja näyttävyyttä.

Rajapinnan suurin hyöty on sen eroavaisuus tavanomaisiin sovellusrajapintoihin. Tavanomaisissa ohjelmarakenteissa sovelluslogiikka ja käyttöliittymä on eroteltu vain luokkakohtaisesti, mutta WPF:ssä niiden erottelu näkyy selvästi. Sovelluksen käyttöliittymän ulkoasu on XAML-merkintäkieltä. Näkyvän käyttöliittymärakenteen takana on sovelluksen bisneslogiikka (Code Behind), joka on eri tiedosto kuin käyttöliittymä. Visual Studiossa sovelluksen rakenne on esitelty kuitenkin järkevästi puurakenteena ja CLR-ajoympäristö yhdistää toteutukset ohjelmaa ajettaessa. WPF:n sovelluslogiikkaa voidaan tehdä usealla .NET-ympäristöön soveltuvalla ohjelmointikielellä kuten C#, VB.NET tai J#.

WPF tukee Microsoftin käyttöjärjestelmiä kuten Windows 7, Windows Vista ja Windows Server 2008, Windows Server 2003 ja on saatavilla Windows XP Service Pack 2:een tai myöhempään versioon. Microsoftin kehittämä Silverlight on web-sovelluksiin käytettävä eräänlainen komponenttikirjaston osajoukko, joka tarjoaa WPF:n kautta Flash-tyylisiä web -ja mobiilisovelluksia.

Tärkein graafinen hyöty WPF:ssä on, että se käyttää Direct3D-moottoria käyttöliittymäkomponenttien mallintamisessa. WPF:n avulla voidaan luoda 2D- sekä 3D-grafiikkaa tai mahdollisesti animaatioita käyttöliittymissä. Nykyään kehitellään paljon sovelluksia, joissa esitellään eri medioita esim. MP3-soitinsovellus. Graafisesti merkittävin ominaisuus on, että WPF tukee vektorigrafiikkaa, joka tarjoaa grafiikan rajattoman skaalautumisen. Skaalautuvuudella tarkoitetaan, että käyttöliittymän komponentteja voidaan suurentaa ja pienentää heikentämättä kuitenkaan niiden tarkkuutta.

Tiedon esittämisessä on näppärä käyttää tiedon sitomista (Data Binding). Sen avulla tieto saadaan näkyviin käyttöliittymässä erilaisia tietotauluja käyttämällä (DataTable). LINQ-kyselykieli on nopea tapa saada yhteys sovelluksesta tietokantaan esim. SQL Server-tai Access-tietokantoihin. WPF tukee myös yleisiä videoformaatteja kuten WMV:tä, MPEG:tä tai AVI:a. [29, s. 1 - 22.]

7.8 LINQ-ohjelmistokomponentti

LINQ ilmestyi .NET Framework versiossa 3.5. Komponentti tarjoaa .NET-kehitykseen aivan uudenlaisen hakuarkkitehtuurin, jota modernit .NET-sovellukset käyttävät. Sen

avulla voidaan hakea tietoa muualtakin kuin relaatiokannoista, esimerkiksi XML-tiedostoista.

LINQ-komponentilla voidaan korvata perinteinen SQL-kielinen kyselysyntaksi. SQL-kielessä ohjelmoijan on muodostettava tietokantahakuja ja näin jo ennalta tunnettava tekniikka. Ohjelmoijan on myös osattava käyttää tarvittavia rajapintoja tietokantahakujen suorittamiseksi. Tästä johtuen ohjelmakoodissa voi olla paljon muuttujia. Tämä estää tarkan virheilmoituksen paikantamisen ohjelmakoodin suorituksen aikana.

Hakuarkkitehtuuri on suunniteltu täysin toimimaan oliokokoelmien kanssa. Hakusyntaksina voidaan käyttää LINQ-kielen omaa syntaksia tai perinteistä SQL-kyselykielen syntaksia. Haut toteutetaan aina oliokokoelmaa vasten, joten kohde täytyy muuttaa ensiksi olioksi. [34.]

7.9 Vektorigrafiikka käyttöliittymässä

7.9.1 Vektorigrafiikan tuomat hyödyt

Vektorigrafiikan rakenne perustuu koordinaatistoon sidottuihin objekteihin, kuten suoriin, monikulmioihin eli polygoneihin, ympyröihin, kaariin jne. Objektien muodot esitetään koordinaatein sekä matemaattisin funktioin. Tämä tekniikka mahdollistaa sen, että vektorigrafiikalla tuotettua kuvan kokoa voidaan muuttaa välttäen rakeisen kuvan muodostumista, toisin kuin perinteistä bittikarttakuvaa suurentamalla voidaan havaita suuriakin pikselivääristymiä.

Kuvan tiedostokoko riippuu kuvassa näkyvien yksityiskohtien perusteella, pikselimäärän sijaan. Tämän ansiosta vektorigrafiikka on resoluutioriippumatonta, ja tallennuskoko on yleensä pienempi kuin bittikarttagrafiikkaa sisältävillä tiedostoilla. Käyttöliittymässä olevien grafiikoiden päivittäminen on tehokkaampaa skaalautuvuuden ansiosta.

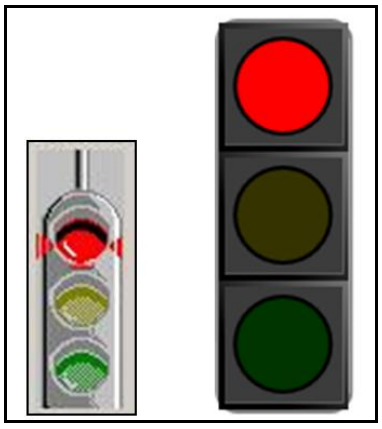
.NET-pohjaiseen varastosovellukseen on toteutettu muutamia vektorigrafiikkaan perustuvia käyttöliittymäkomponentteja. Vanhassa Visual Basic 6 -pohjaisessa sovelluksessa näiden komponenttien grafiikka perustui bittikarttakuviin. VB6-sovellukseen ladataan erikseen kuvat kansiosta, jotka ovat JPEG-formaatissa. Kun taas vektorigrafiikalla piir-

retyt komponentit ovat osa VB.NET-sovelluksen käyttöliittymää. Komponentit näkyvät Visual Studiossa VB.NET-projektin projektipuussa.

Vektorigrafiikkaa käytetään yleisesti 3D-mallien suunnittelussa (CAD) ja piirtografiikan tuottamisessa. Vektorigrafiikka ei kuitenkaan sovellu digitaalisiin valokuviin joissa käytetään pikseligrafiikkaa. Pikseligrafiikkaa ja vektorigrafiikkaa voidaan myös yhdistää, jolloin vektorigrafiikalla luotuja polygoneja pinnoitetaan pikseligrafiikan kuvilla. [35.]

7.9.2 Vektori- ja pikseligrafiikan erot

Muutamassa VB6-pohjaisen sovelluksen lomakkeessa on käytetty pikseligrafiikalla tuotettua grafiikkaa. Varastosovelluksen .NET-pohjaiseen käyttöliittymään nämä grafiikat on osittain korvattu skaalautuvalla vektorigrafiikalla. Aikaisemmassa VB6-pohjaisessa sovelluksessa nosturin statusta esittävä liikennevalo koostuu 6 eri kuvatiedostosta. Tämä johtuu siitä, että jokaiselle liikennevalovälähdykselle on oma tiedostonsa. Esimerkiksi liikennevalossa palaa punainen (kuva 22).



Kuva 22. Liikennevalo bittikartta-ja vektorigrafiikalla toteutettuna (VB.NET)

Vektorigrafiikalla toteutettu liikennevalo .NET-pohjaisessa varastosovelluksessa (kuva 19). Grafiikka on toteutettu Expression Blend 2 -piirtotyökalulla. Komponentti on yksi XAML-päätteinen tiedosto, josta graafiset elementit muodostuvat. Liikennevalon värisävyjä voidaan muuttaa ohjelmallisesti vaihtaen liikennevalo-objektin ellipsimuodon (Ellipse) väriominaisuutta.

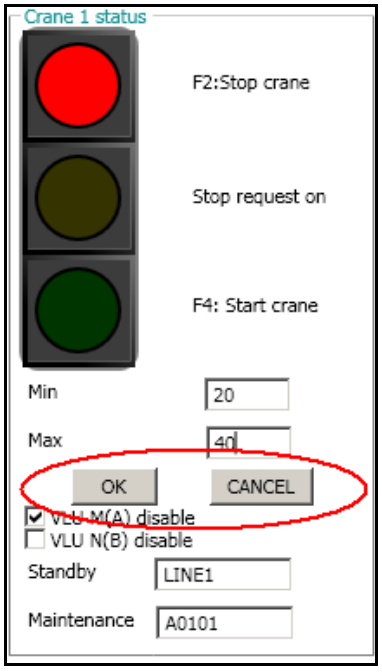
8 .NET-ohjelmointiesimerkkejä ja korjaukset käyttöliittymään

8.1 Dialogin korvaaminen

Aikaisemman VB6-pohjaisen varastosovelluksen dialogeja voidaan vähentää upottamalla dialogi-kysely nykyisen käyttöliittymän sisään. Siitä voidaan tehdä dynaaminen-rakenne, jonka toiminta ohjelmoidaan koodin sisälle.

VB.NET-pohjaisen sovelluksen CraneActivities-välilehdellä on nosturin toimintaa esittävä toiminnallisuus. Jokaisen varastossa olevan nosturin toimintasädettä käyttäjä voi halutessaan muuttaa syöttämällä Min/Max-kenttiin uusia arvoja (kuva 23).

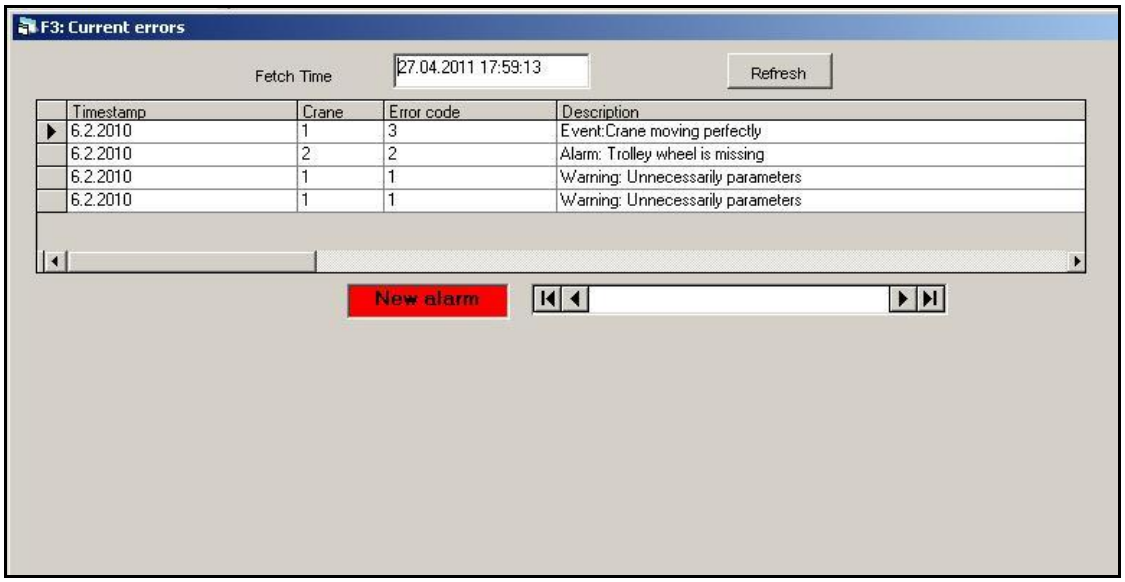
VB6-sovelluksessa käyttöliittymään avautuu Windows-dialogi, jolla toimintasädettä voidaan säätää. Dialogi aktivoituu syötettäessä arvoja Min/Max-kenttiin. Tätä samaa tapahtuman käsittelijää (Event) voidaan käyttää myös VB.NET-sovelluksessa, mutta dialogin sijasta käyttäjä voi syöttää arvot samoihin kenttiin. Vahvistaakseen tämän kontrollien alle ilmestyy hyväksymis- tai hylkäämispainikkeet (liite 1).



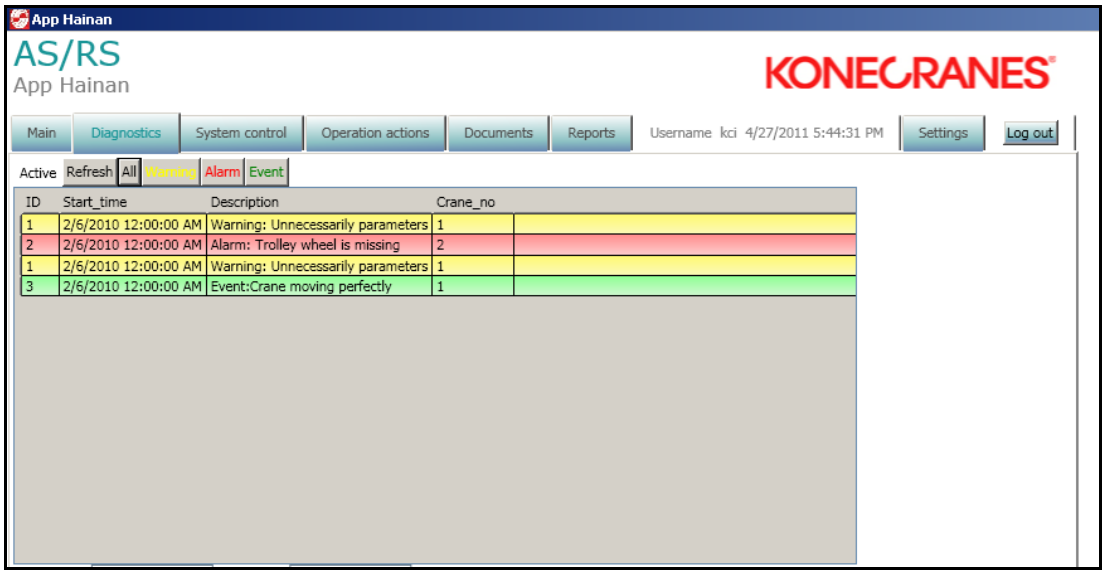
Kuva 23. Nosturin toiminta-alueen muokkaus

8.2 Diagnostiikkataulun värikoodit

Nosturin vikadiagnostiikkaa voidaan seurata VB.NET-pohjaisen sovelluksen käyttöliittymässä olevalta Diagnostics-välilehdeltä (kuva 25). Nosturissa tapahtuneet vikailmoitukset näkyvät tietotauluissa (DataGrid) ja ne on eroteltu väreillä vakavuusasteen mukaan. Aikaisemmassa VB6-sovelluksessa vakavuusasteet on eroteltu vain virhekoodin perusteella (kuva 24). Värikoodatut alueet lisäävät ohjelman käytettävyyttä ja antavat uutta ilmettä käyttöliittymälle. Varastosovelluksen VB.NET-versiossa diagnostiikan ulkoasu on toteutettu XAML-merkintäkielellä (liite 2).



Kuva 24. Vikadiagnostiikkalomake (VB6)

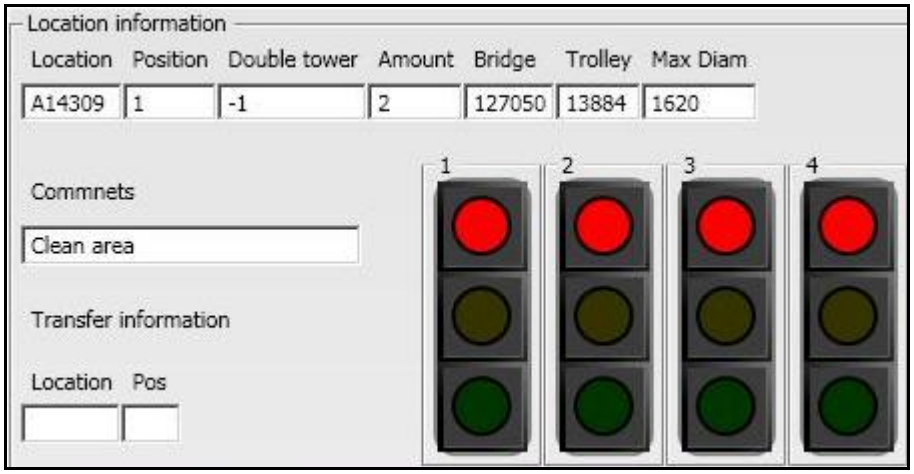


Kuva 25. Vikadiagnostiikan välilehti (VB.NET)

8.3 Liikennevalot

VB.NET-pohjaiseen sovellukseen on toteutettu uudenlaiset nosturin statusta kuvaavat dynaamiset liikennevalot. Liikennevalojen grafiikka perustuu vektorigrafiikkaan ja skaalautuvat aina yhtä tarkasti näytölle. Valojen värejä voidaan säätää tarkasti ja niiden vaihtumiseen voidaan tarvittaessa lisätä myös animaatioita värinvaihdokseen. Ohjelma näyttää käytettävissä olevien nostureiden liikennevalot ja tulostaa liikennevalot tietokannasta saadun tiedon perusteella.

Punainen valo osoittaa, että nosturi on pysähtynyt (kuva 26). Vihreä valo tarkoittaa, että nosturi on liikkeessä. Se voi olla hakemassa paperirullaa tai menossa huoltopaikalle. Keltainen valo tarkoittaa sitä, että käyttäjä on pysäyttänyt nosturin ja nosturi suorittaa vielä viimeisen siirtopyynnön. Jos punainen ja vihreä valo palavat samaan aikaan, niin nosturi vaatii käyttäjän paikalle selvittämään mistä vika aiheutui. Liikennevalon grafiikat on toteutettu XAML-merkintäkielellä (liite 3).



Kuva 26. Liikennevalot varastonäkymässä (VB.NET)

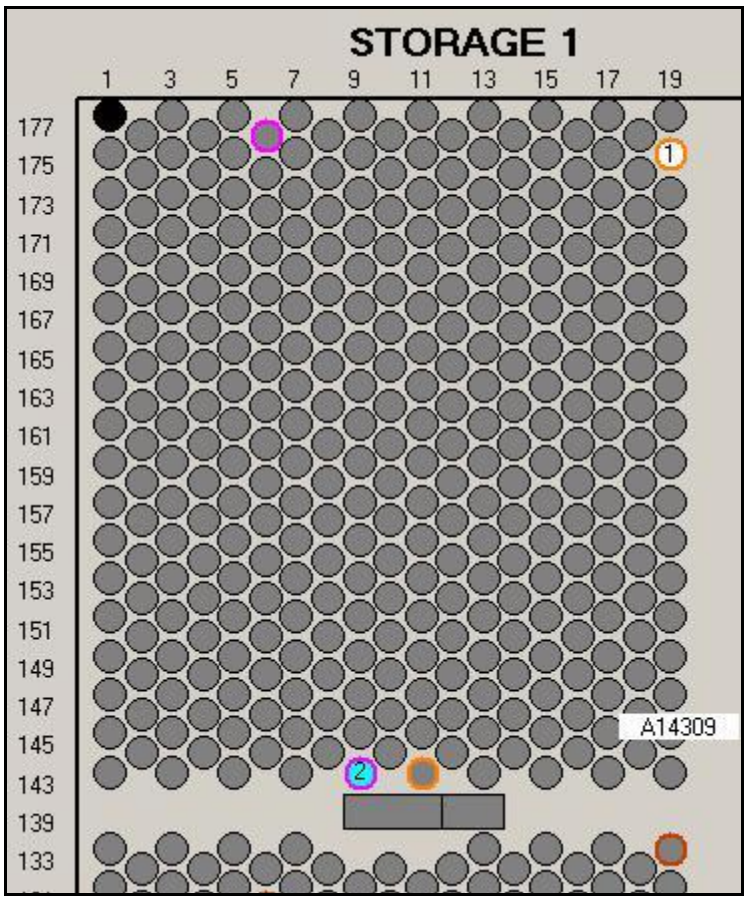
8.4 Paperirullan malli

Paperirullapaikkoja kuvaavat elliptiset bittikarttakuvat on korvattu vektorigrafiikkaan perustuvilla komponenteilla. VB6-pohjaisessa sovelluksessa toiminta perustuu samaan periaatteeseen kuin liikennevaloissa eli kuva ladataan hakemistosta näytölle (kuva 27).

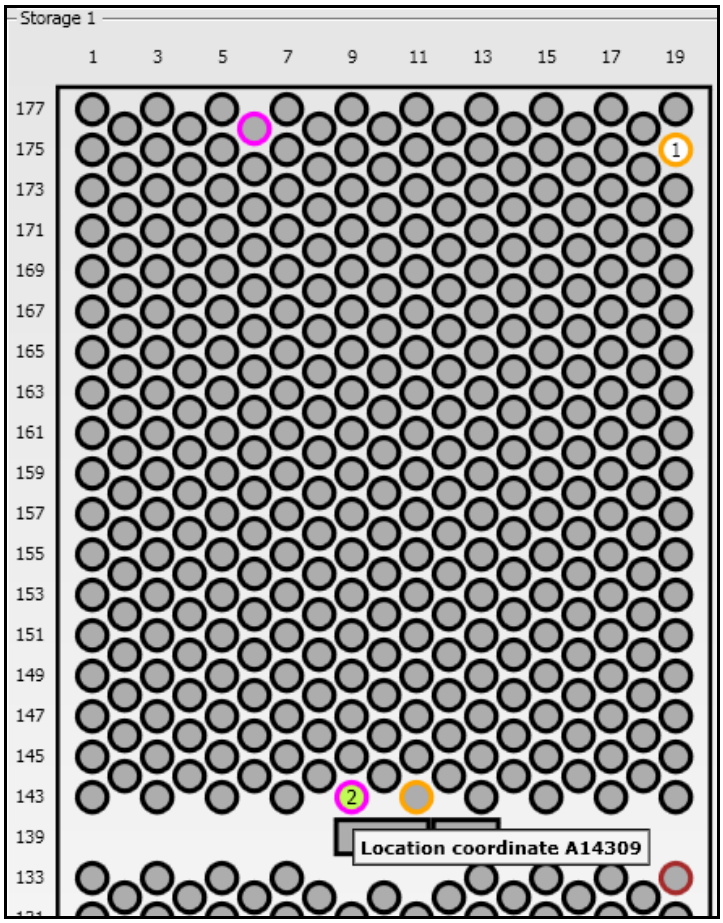
VB.NET-sovelluksen käyttöliittymässä paperirullapaikka on UserControl-tyyppinen luokka. Ulkoapäin katsottuna se näyttää vain graafiselta palikalta, mutta luokka koostuu muutamasta käyttöliittymäkomponentista (kuva 28).

Luokka pitää sisällään tyylit (Style), ruudukon (Grid), elliptisen piirtokomponentin (Ellipse) ja sanomalaatikon (Tooltip).

Valitessaan paperirullapaikan käyttäjä tuplaklikkaa varastonäkymässä olevaa elliptistä piirtokomponenttia. Tämän jälkeen elliptinen alue vaihtaa väriään korostaen valittua paikkaa. Samalla ruutuun ilmestyy sanomalaatikko (ToolTip), joka kertoo paperirullapaikan sijainnin. Tämä operaatio saadaan aikaan elliptisen tapahtuman käsittelijällä (MouseDoubleClick). VB6-sovelluksen käyttöliittymässä elliptinen kuva toimi PictureBox-kontrollin sisällä. PictureBox:lla on kätevä kontrolli, mutta kun yhdistää tyylit, ruudukot ja graafiset komponentit samaan luokkaan niin saadaan paljon monipuolisempi luokka (liite 4).



Kuva 27. Paperirullan malli (VB6)

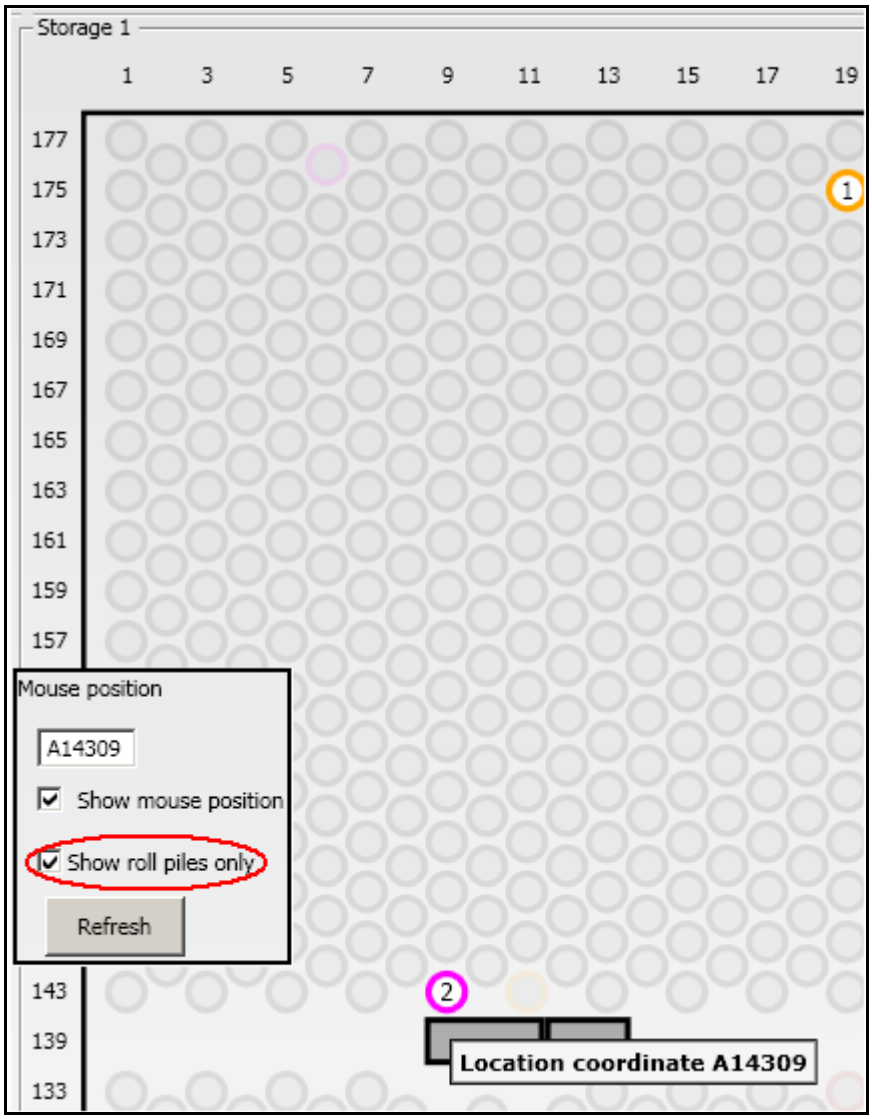


Kuva 28. Paperirullan malli (VB.NET)

8.5 Paperirullapaikan näkyvyystoiminto

VB.NET-sovelluksen käyttöliittymään on lisätty muutamia uusia lisä-toimintoja. Yksi näistä uusista toiminnoista on paperirullien näkyvyyteen liittyvä toiminto. Toiminnon periaate on se, että sen avulla voidaan varastossa näyttää vain ne paperirullapaikat joissa on rullia.

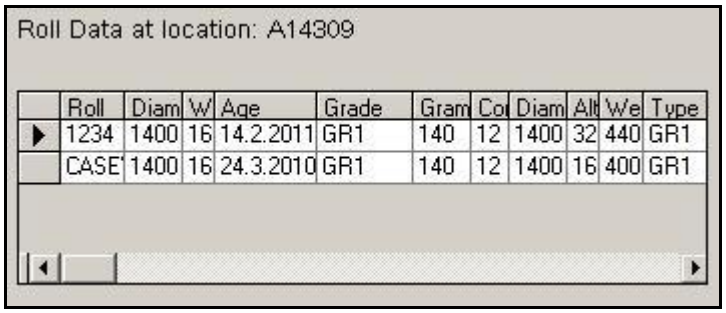
Tässä toiminnossa on hyödynnetty Ellipsi-luokan läpinäkyvyysominaisuutta (Opacity) (kuva 29). Paperirullapaikan läpinäkyvyyttä lisätään mikäli ”Show roll piles only”-tarkastuslaatikko on merkattu (liite 5).



Kuva 29. Paperirullapaikan näkyvyystoiminto (VB.NET)

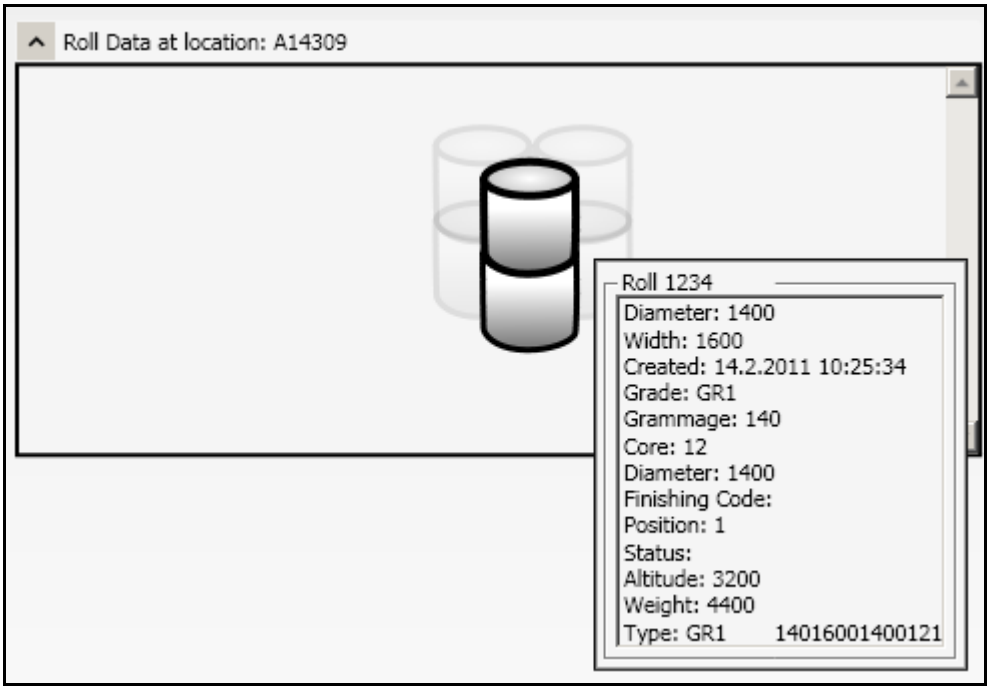
8.6 Paperirullapinon sivuttaisnäkymä

VB6-sovelluksen käyttöliittymässä paperirullatietoja kuvaava taulu (DataTable) on korvattu paperirullapinon sivuttaisnäkymällä. VB6-sovelluksen käyttöliittymässä taulun tiedot ovat luettavissa, mutta se voi olla hankalaa tiedon suuruuden takia. Näin käyttäjä joutuu käyttämään DataTable-kontrollin vierittimiä tiedon lukuun (kuva 30).



Kuva 30. Paperirullan tiedot (VB6)

VB.NET-pohjaisen sovelluksen käyttöliittymässä taulua ei ole lainkaan. Tiedot on upotettu sanomalaatikkoon (ToolTip), joka avautuu käyttäjän siirtäessä hiiren paperirullapinossa olevan paperirullan päälle. Pino luodaan dynaamisesti, ja sen koko riippuu paperirullien määrästä (kuva 31). Etummainen pino kuvaa paperirullapinon todellista kokoa. Taaempana näkyvät himmeämmät alueet kuvaavat vain, että varastossa on muitakin paperirullapinoja. Tällä tavoin saadaan visuaalisempi kuva paperirullavarastossa olevien paperirullien määrästä ja niiden statuksesta (liite 6).



Kuva 31. Paperirullan sivuttaisnäkymä (VB.NET)

8.7 UserControl:n ja Form:n integraatio

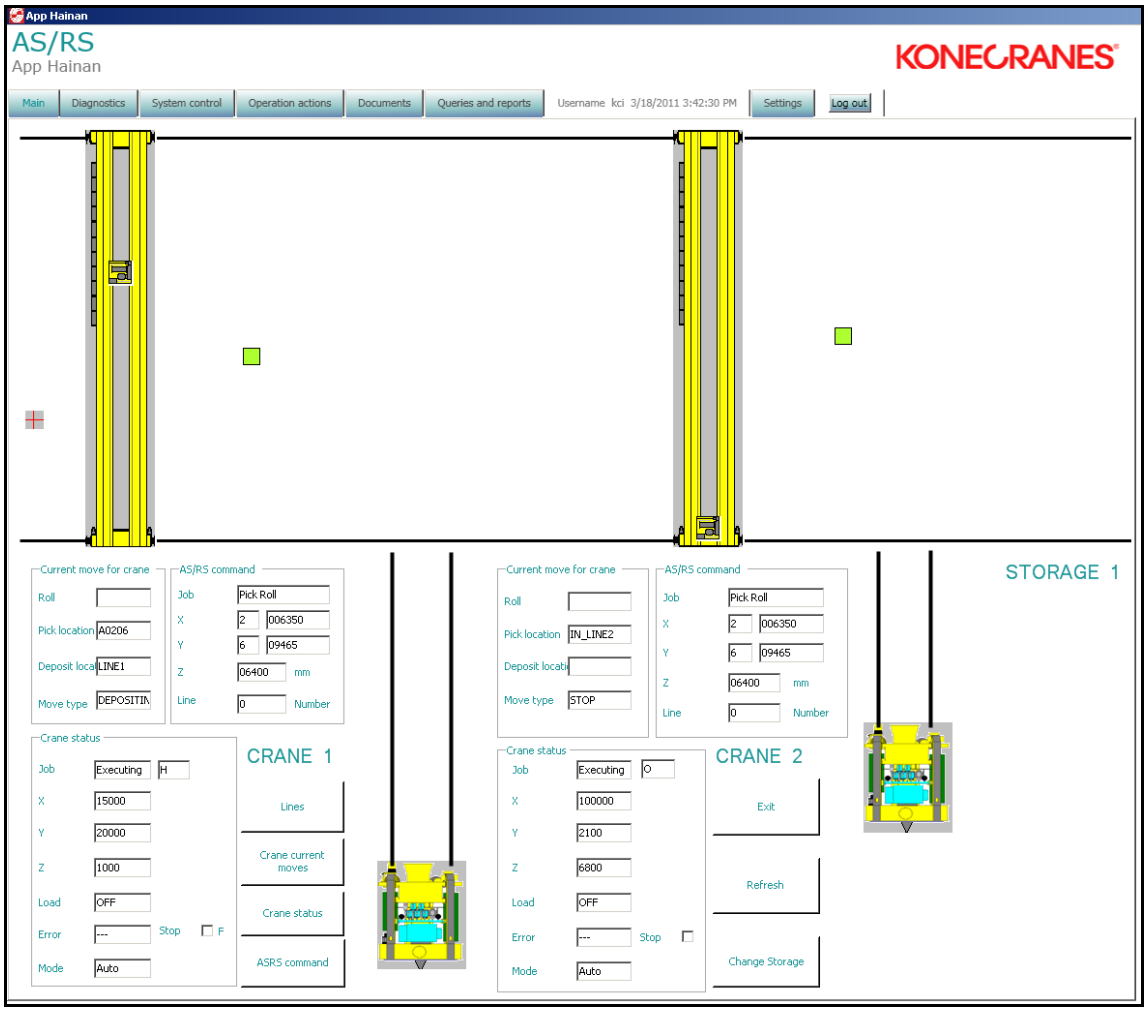
Käännösprosessi on suuri urakka, jos käännettäviä kokonaisuuksia on paljon. VB.NET on hyvä konsepti, sillä se tukee aikaisempia versioita Visual Basic -kielestä. Näin voidaan hyödyntää tätä ominaisuutta ja yhdistää tekniikat yhdeksi kokonaisuudeksi. Kaikkea koodia tai kuvia ei tarvitse välttämättä uudelleen toteuttaa tekniikan yhteensopivuuden vuoksi.

VB.NET-projektiin voidaan lisätä VB6-ohjelmoinnille tuttuja lomakkeita (Form). Lomake voi olla VB.NET:n projektissa erillisenä osana, mutta se on järkevintä yhdistää VB.NET-luokkien kanssa. Lomake voidaan upottaa userControl-luokan sisään sen toimiessa ns. isäntäluokkana. Muutokset tosin täytyy tehdä Form-luokkaan, mutta ei välttämättä kaikkia esimerkiksi käyttöliittymässä voidaan hyödyntää .NET-kontrollien periytymistä. WindowsFormsHost-kontrolli vaatii VB.NET-projektiin WindowsFormsIntegration.dll-sovelluslaajennuksen.

Ohessa XAML-koodi lomakkeen upottamisesta WPF-projektiin:

```
<!—WindowsFormsHost-control represent Form as host-->
            <WindowsFormsHost>
                        <local:frmCMONI></local:frmCMONI>
            </WindowsFormsHost>
```

VB.NET-sovelluksen pääsivulla on nosturinäkymä paperirullavarastosta. Sen avulla käyttäjä voi seurata nosturin sijaintia varastossa. Nosturia kuvaavat keltaiset laatikot ovat bittikarttakuvia. Jos Form-luokka toimii userControl:n lapsena niin userControl:n väriominaisuudet voivat periytyä jokaiseen tekstityyppiseen käyttöliittymäkontrolliin Lomakkeen sisällä (kuva 32).



Kuva 32. Varaston nosturinäkymä (VB.NET)

8.7.1 Twipit pikseleiksi -muunnos

Huomattava käyttöliittymäsuunnitteluun vaikuttava muutos VB6:n ja VB.NET:n välillä on Twipit ja pikselit. VB6-sovelluksen käyttöliittymässä käytetään kontrollien positioinnissa Twip-mittayksikköä. Twipit koostuvat itsenäisistä näytön osakokonaisuuksista näytöllä. Yksi twip on 1/1440 tuumaa tai 17,639 mikrometriä. [2, s. 613.]

VB.NET:ssä sen sijaan käytetään pikseleitä. Se määritellään pisteenä, joka edustaa pienintä graafista elementtiä näytöllä.

Twippejä ja pikseleitä voidaan suhteuttaa keskenään järjestelmien välillä tietyllä suhdeluvulla. Liitteenä on esimerkki, kuinka Twipistä saadaan pikseli (liite 7). Funktio palauttaa pikseliarvon annetulla twip-parametrilla. Muunnosfunktioita on hyvä käyttää .NET-järjestelmissä, jossa on käytetty paljon VB6-ohjelmakoodin Twip-asetuksia kontrolleis-

sa. Visual Studion tarjoamalla migration-työkalulla voidaan kääntää myös vanhan VB6-sovelluksen twip-rakenteita VB6-sovelluslaajennuksen valmiilla VB6.TwipsToPixelsY-funktiolla.

8.7.2 Pikselit twipeiksi -muunnos

Liitteenä on esimerkki, kuinka pikselistä saadaan Twip (liite 7). Funktio palauttaa Twip-arvon annetulla pikseli-parametrilla.

8.8 Kontrollien indeksointi

VB.NET ei tue suoranaisesti staattisesti määriteltyjen kontrollien indeksointia aivan kuten VB6, jossa indeksoinnin voi tehdä kontrollille käyttöliittymää suunnitellessa.

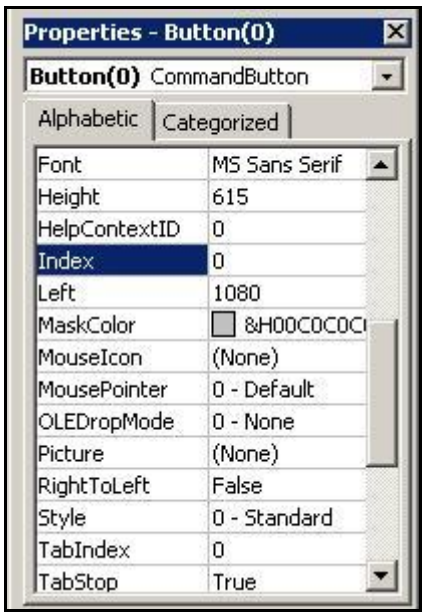
Indeksointi helpottaa kontrollien ominaisuuksien muokkaamista. Jos samanlaisia kontrolleja on lomakkeessa (Form) monta, niin niiden nimi voi olla sama, mutta ne erottuvat indeksien avulla toisistaan. Kun käyttäjä haluaa muokata niiden ominaisuuksia, on helpompi viitata indeksiin kuin jokaiseen kontrolliin erikseen.

Indeksointi yleensä aloitetaan 0:sta ja kasvatetaan kontrollien lisääntyessä. Kontrollin indeksi on ilmoitettu Properties-valikon yläosassa kontrollin nimen yhteydessä. VB6-ohjelmistokehittimen Properties-valikosta indeksoinnin voi tehdä suoraan kontrollille (kuva 33).

VB.NET-projektissa kontrollien indeksointi tehdään hieman eri tavalla kuin VB6:ssa. Ensiksi käyttäjä avaa Visual Studion safe-modessa, mikä tapahtuu käynnistämällä run/suorita johon syötetään teksti ‘devenv /safemode’ ja painetaan OK. Tämän jälkeen vs2008 käynnistyy safe-modessa. Käyttäjä avaa haluamansa projektin tai luo uuden. Sovellus saattaa olla kuitenkin lukittu eikä sitä näin ollen voida näyttää projektipuussa.

VB6-sovelluksen käyttöliittymän Form-lomakkeen kontrollien sijainti, koko ja ominaisuudet on määritelty erilliseen Form-lomakkeen designer-tiedostoon, joka on vb-päätteinen tiedosto. VB.NET-sovelluksen käyttöliittymässä voidaan myös käyttää perinteistä Form-rakennetta, mutta indeksoinnin kontrollin ominaisuuksiin käyttäjä joutuu

itse koodaamaan. Code Behindiin määritetään, mitä tyyppiä indeksi edustaa ja mikä on tietueen arvo (liite 8).

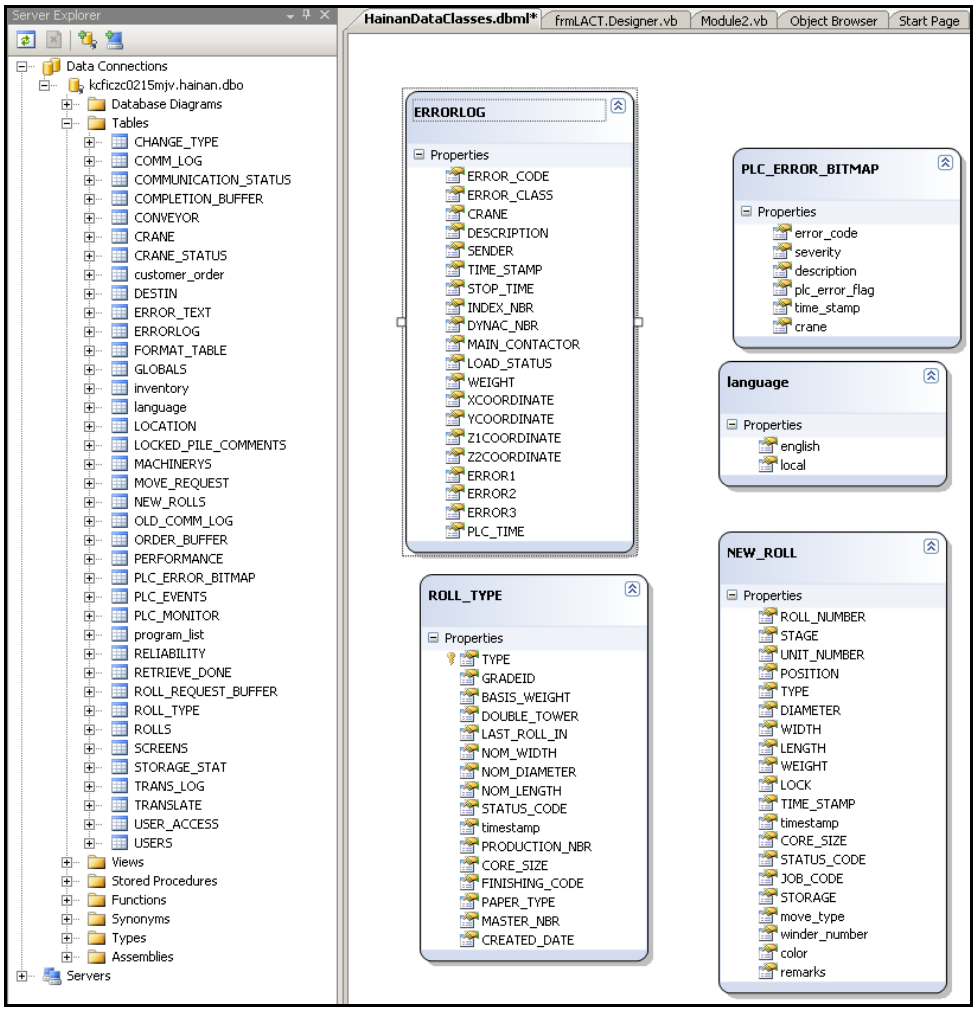


Kuva 33. VB6 kontrollien indeksointi

8.9 LINQ ja tietokantayhteys ADO.NET-rajapintaa käyttäen

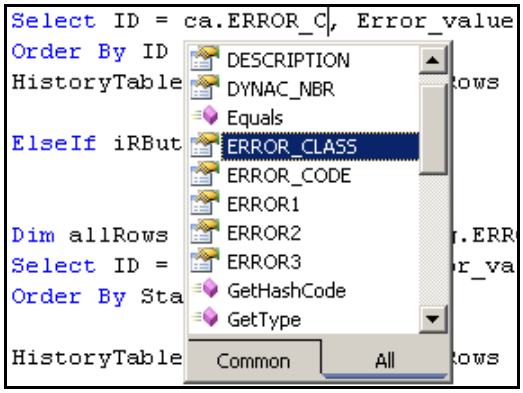
Tietokantayhteyden muodostus VB.NET-sovellukseen voidaan tehdä helposti ADO.NET-tietokantarajapintaa käyttämällä. VB.NET-projektiin voidaan lisätä LINQtoSQL-luokka, joka on mapping-tyyppinen. Se yhdistää tietokannan taulut sekä LINQ:n tarjoamat rajapinnat tietokannan käsittelylle.

Luokka generoi tietokannasta saadun tiedon IEnumerable-pohjaiseksi olio-kokoelmaksi. Tämän jälkeen käyttäjä voi raahata haluamansa taulut projektiin, joista tiedon voi myöhemmin hakea (kuva 34). [36.]



Kuva 34. Tietokantataulut ADO.NET:ssä (VB.NET)

Päästääkseen käsiksi IEnumerable-kokoelmaan käyttäjän on luotava instanssi määrittelemästään LINQtoSQL-luokasta (liitteet 2; 9). Tämän jälkeen kokoelmaluokasta löytyy taulun tiedot, joihin käyttäjä pääsee käsiksi LINQ-kyselykielellä. Tietokantahakuihin voi käyttää apuna Visual Studion intellisense-toimintoa, joka listaa taulun attribuutit annettujen kirjainten perusteella (kuva 35).



Kuva 35. Intellisense listaus (VB.NET)

9 Tulosten pohdintaa ja jatkotoimenpiteet

Tuloksista voidaan todeta, että osa varastosovelluksen toiminnoista on uudenaikaistettu VB.NET-tekniikkaan perustuen. Käyttöliittymään liittyvistä toiminnoista osa toteutettiin XAML-merkintäkielellä, joka näkyy grafiikan tarkkuudessa ja värimaailmassa. Sen avulla esimerkiksi käyttöliittymässä esiintyvät liikennevalot saivat täysin uutta ilmettä vektorigrafiikan myötä. VB.NET-pohjaisessa varastosovelluksessa ei käytetä VB6-version bittikarttakuvia, vaan ne on korvattu uusilla käyttöliittymärakenteilla.

Varastonäkymän ja pääsivun toiminnot ovat entistä johdonmukaisempia, kun ei tarvitse klikata hiirellä montaa kertaa. Vanhassa VB6-sovelluksessa tämä toiminnallisuus aiheutti hieman päänvaivaa. Vikadiagnostiikkanäkymän taulun ulkoasu korvattiin VB.NET:n tarjoamilla kontrolleilla ja uuteen VB.NET-versioon lisättiin värikoodit. Molempien tekniikoiden yhteensopivuus näkyy hyvin tekniikoiden integraationa. Integraatio voi olla hyödyllinen, jos käyttöliittymään ei haluta graafisia muutoksia. Tämä voi nopeuttaa ohjelmistomigraation etenemistä huomattavasti.

VB.NET-version jatkotoimenpiteet jatkuvat testauksen parissa. Insinöörityön edistyessä testit ovat rajoittuneet vain varastosovelluksen sisäisiin yksikkö ja moduulitestaukseen. Myöhemmässä vaiheessa kun varastosovelluksen VB.NET-versio on kokonaan valmis, siirrytään järjestelmätestaukseen. Siinä testataan sovellusta varastonhallintajärjestelmän kanssa, joka on varastosovelluksen pohja. Tässä vaiheessa testataan jo pystyrullavaraston nostureilta välittyviä viestejä tietokantaan ja sitä kautta varastosovellukseen.

Muutamia toimintoja VB.NET-versiossa on vielä kesken, mutta sovellus toimii jo moitteettomasti. Siihen lisätään vielä ainakin Reporting service -linkitys, jonka avulla tulostetaan nosturin diagnostiikkaa tulostimelle. Reporting service -toiminto liittyy varastoraportointiin, jonka avulla prosessikäyttäjä voi pitää kirjaa pystyrullavarastossa tapahtuvista toiminnoista. Tähän asti totutetut toiminnot ja kokonaisuudet muodostavat jo hyvässä kehitysvaiheessa olevan varastosovelluksen, jota käytetään yrityksen tulevissa kehitysprojekteissa pohjana.

10 Yhteenveto

Päätavoitteena työssä oli kääntää Visual Basic 6 -tekniikalla toteutettu varastosovellus Visual Basic.NET -pohjaiseksi sovellukseksi ja esitellä tekniikoita, joilla käännösprosessi toteutettiin. Jos insinöörityössä käsiteltäisiin yhtä perinpohjaisesti käännösprosessia kuin osassa lähteitä, niin työ olisi erittäin laaja. Tavoitteena olikin saada mahtumaan työhön ne asiat, jotka ovat tärkeitä käännösprosessia toteuttaessa.

Insinöörityön alussa käytettiin jonkun verran aikaa siihen, kuinka ohjelmistomigraatiota kannattaisi lähteä toteuttamaan ja miten siitä syntyneestä VB.NET-pohjaisesta sovelluksesta saataisiin kilpailukykyinen ohjelmistotuote yrityksen muiden järjestelmien rinnalla. Työn alussa tehtiin paljon muistiinpanoja siitä, miten käännösprosessi voisi edetä ja mitä uusia ominaisuuksia uuteen varastosovellukseen kannattaisi toteuttaa. Muistiinpanot osoittautuivat hyväksi ideaksi insinöörityön valmisteluun. Muistikirjaan hahmoteltiin hieman käännösprosessin vaiheita ja lueteltiin myös lähdeviitteitä.

Insinöörityö onnistui alkuperäisessä tavoitteessaan. Työn tuloksena valmistui uusi versio varastosovelluksesta Visual Basic.NET -tekniikkaan perustuen. VB.NET-pohjaisessa sovelluksessa on samat toiminnot kuin aikaisemmallakin Visual Basic 6 -versiolla. Tämän lisäksi siihen toteutettiin muutamia lisäominaisuuksia, joiden käytettävyydestä on saatu hyvää palautetta. Ohjelmallisten toimintojen ohella keskityttiin myös ohjelman käyttöliittymäsuunnitteluun, mikä avasi uusia näkökulmia ohjelmistoprojektin tekoon. Projektia jatketaan tämän jälkeen ohjelman käyttöliittymäsuunnittelulla ulkoisten toimijoiden avustuksella. Varastosovellus toimii eräänlaisena prototyyppinä tuleville kehitysprojekteille.

Lähteet

1 Visual Basic, Wikipedia. 2011. Verkkodokumentti. <http://fi.wikipedia.org/wiki/Visual_Basic> Luettu 28.02.2011.

2 Visual Basic Ohjelmoijan käsikirja.Microsoft Press 1998.

3 Microsoft 2005, Upgrading Visual Basic 6.0 Applications to Visual Basic .NET and Visual Basic 2005, EBOOK < http://www.microsoft.com/downloads/en/details.aspx?FamilyId=7C3FE0A9-CBED-485F-BFD5-847FB68F785D&displaylang=en > Luettu 28.12.2010.

4 Software Branding, Microsoft 2011.Verkkodokumentti. <http://msdn.microsoft.com/en-us/library/aa511284.aspx> Luettu 2.2.2011.

5 Introducing the Architect. Microsoft 2011.Verkkodokumentti. <http://msdn.microsoft.com/en-us/library/aa302162.aspx> Luettu 4.2.2011.

6 .NET Framework Conceptual Overview, Microsoft 2011.Verkkodokumentti. <http://msdn.microsoft.com/library/zw4w595w.aspx> Luettu 7.3.2011.

7 Common Language Runtime Overview, Microsoft 2011.Verkkodokumentti. <http://msdn.microsoft.com/en-us/library/ddk909ch%28v=vs.71%29.aspx> Luettu 11.3.2011.

8 .NET Framework, Wikipedia. 2011. Verkkodokumentti. <http://en.wikipedia.org/wiki/.NET_Framework> Luettu 7.3.2011.

9 Microsoft Visual Studio, Wikipedia. 2011. Verkkodokumentti. <http://en.wikipedia.org/wiki/Microsoft_Visual_Studio> 8.3.2011.

10 Expression Blend, Wikipedia. 2011. Verkkodokumentti. <http://en.wikipedia.org/wiki/Microsoft_Expression_Blend> 8.3.2011.

11 Microsoft SQL Server 2008 R2 Unleashed. Ray Rankins, Paul Bertucci, Chris Gallelli ja Alex T. Silverstein. SAMS 2011. Ebook <http://uploading.com/files/get/db9c181m/> Luettu 8.3.2011.

12 Upgrading Visual Basic 6.0 Applications to Visual Basic .NET and Visual Basic 2005, Microsoft 2011.Verkkodokumentti. <http://msdn.microsoft.com/en-us/library/aa480541.aspx> Luettu 9.3.2011.

13 10 steps to migrate existing code to VB.NET. 2011. Verkkodokumentti. <http://www.aivosto.com/vbtips/vbnetmigration.html> Luettu 11.3.2011.

14 Visual Basic Naming Conventions.Microsoft 2011. Verkkodokumentti. <http://msdn.microsoft.com/en-us/library/0b283bse%28v=vs.71%29.aspx> Luettu 11.3.2011.

15 VB.NET compatibility rules, Aivosto Oy 2011. Verkkodokumentti. <http://www.aivosto.com/project/help/enterprise-netcheck-rules.html> Luettu 11.3.2011.

16 ADO.NET. MSDN 2011. Verkkodokumentti <http://msdn.microsoft.com/en-us/library/aa286484.aspx> Luettu 11.3.2011.

17 Differences Between Visual Basic 6.0 and .NET Control, Microsoft 2001. Verkkodokumentti <http://msdn.microsoft.com/en-us/library/ms973208.aspx> Luettu 14.3.2011.

18 Upgrading Microsoft Visual Basic 6.0 to Microsoft Visual Basic.NET. Ed Robinson, Michael Bond, Ian Oliver. Microsoft Press 2002.EBOOK.

19 TabControl Class.Microsoft 2011.Verkkodokumentti <http://msdn.microsoft.com/en-us/library/system.windows.controls.tabcontrol%28v=VS.95%29.aspx> Luettu 14.3.2011.

20 Sean Sexton, 2010. Varkkodokumentti <http://wpf.2000things.com/tag/wpf-control-hierarchy/> luettu 28.12.2010.

21 Controls and Programmable Objects Compared in Different Languages and Libraries.Microsoft 2011.Verkkodokumentti <http://msdn.microsoft.com/en-us/library/0061wezk%28v=vs.71%29.aspx > Luettu 14.3.2011.

22 Marc Israel, July 5, 2000.Verkkodokumentti <http://www.databasejournal.com/features/mssql/article.php/1490571/From-DAO-to-ADO.htm> Luettu 29.12.2010.

23 Data Type Summary, Microsoft 2010. Verkkoaineisto <http://msdn.microsoft.com/en-us/library/47zceaw7%28v=vs.71%29.aspx> Luettu 29.12.2010.

24 Using IntelliSense.Microsoft 2011. Verkkodokumentti <http://msdn.microsoft.com/en-us/library/hcw1s69b%28v=VS.80%29.aspx> Luettu 15.3.2011.

25 Improving usability. Microsoft 2010. Verkkodokumentti <http://msdn.microsoft.com/en-us/library/aa511329.aspx > Luettu 29.12.2010.

26 Advantages of VB.NET .Verkkodokumentti <http://dev.fyicenter.com/Interview-Questions/dotNet-1/Advantages_of_VB_NET.html > Luettu 29.12.2010.

27 What is XAML? Verkkodokumentti 2010. <http://msdn.microsoft.com/en-us/library/ms752059.aspx> Luettu 27.12.2010.

28 Lori A. MacVittie, O'Reilly 2006 XAML in a Nutshell.

29 Matthew MacDonald, APRESS 2008 Pro WPF with VB 2008: Windows Presentation Foundation with .NET 3.5.

30 Jared Proudfoot 16.4.2008.Verkkodokumentti. <http://blogs.technet.com/b/lifecycle/archive/2008/04/16/end-of-support-for-visual-basic-6-0.aspx> Luettu 30.12.2010.

31 ADO.NET.Microsoft 2011. Verkkodokumentti. <http://msdn.microsoft.com/en-us/library/aa286484.aspx> Luettu 30.12.2011.

32 ADO.NET Chapter 5. Verkkodokumentti <http://etutorials.org/Programming/.NET+Framework+Essentials/Chapter+5.+Data+and+XML/5.2+ADO.NET+Benefits/> Luettu 30.12.2010.

33 Juhani Kyllönen 2005 Opinnäytetyö MICROSOFT ADO.NET OHJELMISTOTUOTANNOSSA.

34 LINQ: .NET Language-Integrated Query. MSDN 2011. Verkkodokumentti. <http://msdn.microsoft.com/library/bb308959.aspx> Luettu 15.3.2011.

35 Ira Greenberg. Vector Graphics, Wikipedia 2011. Verkkodokumentti <http://en.wikipedia.org/wiki/Vector_graphics 2007> Luettu 15.3.2011.

36 LINQ and ADO.NET. Microsoft 2011. Verkkodokumentti. <http://msdn.microsoft.com/en-us/library/bb399365.aspx> Luettu 18.3.2011.

Dialogin korvaaminen

UserControlAMONI.xaml.vb

```
Private Sub pbOkOperatingValues(ByVal sender As System.Object, ByVal e
As System.Windows.RoutedEventArgs)

        'Function will activate when user push the OK-button
        '(Click-event)

        'Local variables
        Dim cmdTemp As New ADODB.Command
        Dim tmpBtn As Button
        tmpBtn = sender
        Dim sMin, sMax As String
        Dim craneNum As Short
        craneNum = CShort(tmpBtn.Tag)

        'Set new values to variables from textboxes
        sMin = trafficLightCompArray(craneNum).tbMin.Text
        sMax = trafficLightCompArray(craneNum).tbMax.Text
        'Crane number 1,2,3 or 4
        craneNum = craneNum + 1
        Try

        'Parameter checking. Numeric parameters only allowed
        If Not IsNumeric(sMin) Or Not IsNumeric(sMax) Then

            MsgBox("Please input a numeric value!!!")

            Else
            'Activated connection is opened and new updates executed
            cmdTemp.ActiveConnection = AdoCon
            cmdTemp.CommandText = "UPDATE crane set operating_min =" &
sMin & ", operating_max =" & sMax & " where crane_number =" &
craneNum.ToString
            cmdTemp.Execute()
            cmdTemp = Nothing

        End If

        craneNum = craneNum - 1

        'Minimize the area, where OK and CANCEL are placed
        trafficLightCompArray(craneNum).grUpdateOperatingValues.Height
= 0.0

trafficLightCompArray(craneNum).grUpdateOperatingValues.Visibility =
Windows.Visibility.Hidden

        'If there is some exception in progress then message has given
        Catch ex As Exception
            MessageBox.Show(ex.ToString)
        End Try


End Sub
```

```
Private Sub pbCancelOperatingValues(ByVal sender As System.Object,
ByVal e As System.Windows.RoutedEventArgs)
        'Function will activate when user push the CANCEL-button
        '(Click-event)
        'Purpose of this function is cancel modifications made by user
        'Local variables
        Dim pbTmp As Button
        pbTmp = sender
        Dim craneNum As Short
        craneNum = CShort(pbTmp.Tag)
        Dim rstTemp As New ADODB.Recordset
        Dim qStr As String
        'Crane number 1,2,3 or 4
        craneNum = craneNum + 1
        Try

        'Activated connection is open
        qStr = "select operating_min,operating_max from crane where
crane_number=" & craneNum.ToString
        rstTemp.let_ActiveConnection(AdoCon)
        rstTemp.Open(qStr)

        With rstTemp.EOF
            'Sets values from database to the variables
            operatingValMin = rstTemp.Fields(0).Value
            operatingValMax = rstTemp.Fields(1).Value

        End With
        craneNum = craneNum - 1
        'Set values of variables to the textboxes
        trafficLightCompArray(craneNum).tbMin.Text = operatingValMin
        trafficLightCompArray(craneNum).tbMax.Text = operatingValMax
           'Minimize the area, where OK and CANCEL are placed
        trafficLightCompArray(craneNum).grUpdateOperatingValues.Height
= 0.0

trafficLightCompArray(craneNum).grUpdateOperatingValues.Visibility =
Windows.Visibility.Hidden
        rstTemp.Close()
        rstTemp = Nothing
        'If there is some exception in progress then message has given
Catch ex As Exception

        End Try


End Sub
```

```
Private Sub fOperatingVal(ByVal sender As System.Object, ByVal e As
System.Windows.RoutedEventArgs)

        'Function will activate when textbox's text changed
        'Local variables
        Dim cbTemp As TextBox
        cbTemp = sender
        Dim craneNum As Short
        Dim dbCraneN As Short
        craneNum = CShort(cbTemp.Tag)
        dbCraneN = craneNum + 1

        'If the crane has stopped then new values are allowed to set
        'Minimize the area, where OK and CANCEL are placed
        If IsCraneStopped(dbCraneN) = True Then

trafficLightCompArray(craneNum).grUpdateOperatingValues.Visibility =
Windows.Visibility.Visible

trafficLightCompArray(craneNum).grUpdateOperatingValues.Height = 22
        Else
            MsgBox("Stop the crane first!!!")
        End If

End Sub
```

UserControlCraneLightComp.xaml

```
<!--Ok and Cancel button specification in XAML. Grid describes the
area where buttons are placed. Grid is invisible ja height=0-->
                    <Grid Visibility="Hidden"
Name="grUpdateOperatingValues" Height="0">

                            <StackPanel Orientation="Horizontal">
                                <Button Width="50" Height="22"
Margin="30,0,0,0" Name="pbUpDateOperationValOK" Content="OK">
                            </Button>
                                <Button Width="60" Height="22"
Margin="30,0,0,0" Name="pbUpdateOperationValCANCEL"
Content="CANCEL"></Button>
                            </StackPanel>

                    </Grid>
```

Diagnostiikkataulun värikoodit

UserControlLOGS.xaml.vb

```
'Instance of data entity
Private errlog As New HainanDataClassesDataContext
Private sRButtonstate As String = ""
Private iRButtonState As Integer = 0


Private Sub All_Click(ByVal sender As System.Object, ByVal e As
System.Windows.RoutedEventArgs)
    'Order by time or ID despite the error class of the table
    'Function uses LINQ-syntax its database queries
    rButtonState()

    If dateIsOn() = False Then

    If iRButtonState = 1 Then

        Dim allRows = From ca In errlog.ERRORLOGs _
        Select ID = ca.ERROR_CODE, Error_value = ca.ERROR_CLASS,
Start_time = ca.TIME_STAMP, Description = ca.DESCRIPTION, Crane_no =
ca.CRANE _
        Order By ID
        HistoryTable.ItemsSource = allRows

        ElseIf iRButtonState = 2 Then


        Dim allRows = From ca In errlog.ERRORLOGs _
        Select ID = ca.ERROR_CODE, Error_value = ca.ERROR_CLASS,
Start_time = ca.TIME_STAMP, Description = ca.DESCRIPTION, Crane_no =
ca.CRANE _
        Order By Start_time

        HistoryTable.ItemsSource = allRows

        Else

        Dim allRows = From ca In errlog.ERRORLOGs _
        Select ID = ca.ERROR_CODE, Error_value = ca.ERROR_CLASS,
Start_time = ca.TIME_STAMP, Description = ca.DESCRIPTION, Crane_no =
ca.CRANE

        HistoryTable.ItemsSource = allRows

    End If
Else
        orderBySelectedDate()

End If

End Sub
```

UserControlLOGS.xaml.vb

```
<UserControl x:Class="UserControlLOGS"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:dg="http://schemas.microsoft.com/wpf/2008/toolkit"
    xmlns:forms="clr-
namespace:System.Windows.Forms;assembly=System.Windows.Forms"
    xmlns:errorlog="clr-namespace:APP_Hainan"
    Loaded="ucLOGS_Loaded"     Width="auto" Height="auto">

    <UserControl.Resources>
        <!--Defined datacontext-->
        <errorlog:HainanDataClassesDataContext
x:Key="ErrorLog"></errorlog:HainanDataClassesDataContext>
        <!--Style specifications for the crane faults.-->
        <Style x:Key="RowStyle" TargetType="{x:Type dg:DataGridRow}">
            <Style.Resources>
               <LinearGradientBrush x:Key="WarningBrush"
EndPoint="0.5,1" StartPoint="0.5,-0.075">
                   <GradientStop Color="#FFFBF9BB" Offset="1"/>
                   <GradientStop Color="#FFFFF96B" Offset="0"/>
                </LinearGradientBrush>
                <LinearGradientBrush x:Key="AlertBrush"
EndPoint="0.5,1" StartPoint="0.5,-0.075">
                   <GradientStop Offset="1.0" Color="#FFFDCECE" />
                   <GradientStop Offset="0" Color="#FFFF8989" />
                </LinearGradientBrush>
                <LinearGradientBrush x:Key="EventBrush"
EndPoint="0.5,1" StartPoint="0.5,-0.075">
                   <GradientStop Color="#FFD2F7D3" Offset="1"/>
                   <GradientStop Color="#FF8AFF8E" Offset="0"/>
                </LinearGradientBrush>
                <errorlog:ErrorColorTemplateSelector
x:Key="ErrorColorConv"></errorlog:ErrorColorTemplateSelector>
            </Style.Resources>
            <Setter Property="HorizontalContentAlignment"
Value="Stretch" />
            <Style.Triggers>
                <!--Trigger get answer from ErrorColorConv-class which
setter apply and setter
                sets a specific value to row's background-->
                <DataTrigger Binding="{Binding Path=Description,
Converter={StaticResource ErrorColorConv}}" Value="1">
                   <Setter Property="Background"
Value="{StaticResource WarningBrush}"/>
                </DataTrigger>

                <DataTrigger Binding="{Binding Path=Description,
Converter={StaticResource ErrorColorConv}}" Value="2">
                   <Setter Property="Background"
Value="{StaticResource AlertBrush}"/>
                </DataTrigger>

                <DataTrigger Binding="{Binding Path=Description,
Converter={StaticResource ErrorColorConv}}" Value="3">
```

```
                    <Setter Property="Background"
Value="{StaticResource EventBrush}"/>
                </DataTrigger>
            </Style.Triggers>
        </Style>
        <Style x:Key="datePickerStyle" TargetType="{x:Type
dg:Calendar}">

            <Setter Property="Background" Value="White"></Setter>
            <Setter Property="Language" Value="en-US"></Setter>
        </Style>
    </UserControl.Resources>
    <Grid>
        <StackPanel Orientation="Vertical">
            <Grid>
                <StackPanel Orientation="Vertical">
                    <Grid>
                        <StackPanel Orientation="Horizontal">
                            <Label Name="lblActive">Active</Label>
                            <Button Click="RefreshHistory_Click"
Name="RefreshHistory">Refresh</Button>
                            <Button Name="btnAllAct"
Click="All1_Click_Click">All</Button>
                            <Button Name="btnWarningAct"
Foreground="Yellow" Click="Warning1_click_Click">Warning</Button>
                            <Button Name="btnAlarmAct"
Foreground="Red" Click="Alarm1_click_Click">Alarm</Button>
                            <Button Name="btnEventAct"
Foreground="Green" Click="Event1_click_Click">Event</Button>
                        </StackPanel>
                    </Grid>
                    <dg:DataGrid ItemContainerStyle="{StaticResource
RowStyle}" ItemsSource="{Binding}" Height="300" Width="670"
Name="ActiveTable" HorizontalAlignment="Left"
AutoGenerateColumns="True">
                    </dg:DataGrid>
                    <Grid>
                        <StackPanel Orientation="Horizontal">
                            <Label Name="lblStartDate">Start date:
</Label>

                            <dg:DatePicker
CalendarStyle="{StaticResource datePickerStyle}"
Name="StartLogsDatePicker">

                            </dg:DatePicker>
                            <Label Name="lblStopDate">Stop date:
</Label>
                            <dg:DatePicker
CalendarStyle="{StaticResource datePickerStyle}"
Name="StopLogsDatePicker"></dg:DatePicker>
                        </StackPanel>
                    </Grid>
                    <RadioButton Name="errByTime_RButton"
Content="Order by time"></RadioButton>
                    <RadioButton Name="errByID_RButton" Content="Order
by ID"></RadioButton>
                </StackPanel>
```

```
            </Grid>
            <Grid>
                <StackPanel Orientation="Horizontal">
                    <Grid>
                        <Label Name="lblLOGS">Diagnostics</Label>
                    </Grid>
                    <Label Name="lblSHOW">Show:</Label>
                    <Button Width="60" Name="btnAllHist"
Click="All_Click">All</Button>
                    <Button Width="60" Name="btnWarningHist"
Foreground="Yellow" Click="Warning_Click">Warning</Button>
                    <Button Width="60" Name="btnAlarmHist"
Foreground="Red" Click="Alarm_Click">Alarm</Button>
                    <Button Width="60" Name="btnEventHist"
Foreground="Green" Click="Event_Click">Event</Button>
                    <TextBox Width="60" Name="itemCheck"></TextBox>
                </StackPanel>
            </Grid>
            <Label Name="lblHistory">History</Label>
            <dg:DataGrid ItemContainerStyle="{StaticResource
RowStyle}" ItemsSource="{Binding}" Height="300" Width="670"
Name="HistoryTable" HeadersVisibility="Column"
HorizontalAlignment="Left">
            </dg:DataGrid>
        </StackPanel>
    </Grid>
</UserControl>
```

ErrorColorTemplateSelector.vb

```
Imports System.Data
Imports System.Diagnostics
Imports System.Globalization
Imports System.Windows.Data
Imports System.Windows.Media

' This class returns a numeric value for colortemplate trigger in
usercontrol class
' indicates if it is "War", "Ala", or "Eve".
' Returns 1, 2, or 3.

<ValueConversion(GetType(Object), GetType(Integer))> _
Public Class ErrorColorTemplateSelector
    Implements IValueConverter

        Public Function Convert(ByVal value As Object, ByVal
targetType As System.Type, ByVal parameter As Object, ByVal culture As
System.Globalization.CultureInfo) As Object Implements
System.Windows.Data.IValueConverter.Convert

            Dim number As String =
CStr(System.Convert.ChangeType(value, TypeCode.String))

             If number.Contains("War") Then
                Return 1
            End If

            If number.Contains("Ala") Then
                Return 2
            End If

            If number.Contains("Eve") Then
                Return 3
            End If

            Return 2

        End Function

        Public Function ConvertBack(ByVal value As Object, ByVal
targetType As System.Type, ByVal parameter As Object, ByVal culture As
System.Globalization.CultureInfo) As Object Implements
System.Windows.Data.IValueConverter.ConvertBack
            Throw New NotSupportedException("Convert back is not
supported!!")
        End Function
End Class
```

Liikennevalot

UserControlTrafficLightBorder.xaml

```
<UserControl x:Class="UserControlTrafficLightBorder"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Width="auto" Height="auto">
    <Grid>
        <Grid>
            <GroupBox IsEnabled="False"  x:Name="trafficLightGB">
            <Border Background="LightGray"
                    CornerRadius="10"
                    BorderBrush="Gray"
                    BorderThickness="2">
                <StackPanel VerticalAlignment="Center">
                    <StackPanel.Resources>
                    <!--Style resources of the lights-->
                        <Style x:Key="df" TargetType="{x:Type
Ellipse}">
                            <Setter Property="Width" Value="30" />
                            <Setter Property="Height" Value="30" />
                            <Setter Property="Fill" Value="Red" />
                            <Setter Property="Stroke" Value="Black" />
                            <Setter Property="StrokeThickness"
Value="2" />
                            <Setter Property="Margin" Value="4" />
                        </Style>
                        <Style x:Key="trafficLightYellow"
TargetType="{x:Type Ellipse}">
                            <Setter Property="Width" Value="30" />
                            <Setter Property="Height" Value="30" />
                            <Setter Property="Fill" Value="Yellow" />
                            <Setter Property="Stroke" Value="Black" />
                            <Setter Property="StrokeThickness"
Value="2" />
                            <Setter Property="Margin" Value="4" />
                        </Style>
                        <Style x:Key="trafficLightGreen"
TargetType="{x:Type Ellipse}">
                            <Setter Property="Width" Value="30" />
                            <Setter Property="Height" Value="30" />
                            <Setter Property="Fill" Value="DarkGreen"
/>
                            <Setter Property="Stroke" Value="Black" />
                            <Setter Property="StrokeThickness"
Value="2" />
                            <Setter Property="Margin" Value="4" />
                        </Style>
                    </StackPanel.Resources>
                    <Button Name="pbStopCrane" Background="#FF3C3C3C"
Foreground="#FF000000">
                        <Button.BorderBrush>
                            <LinearGradientBrush
EndPoint="1.219,0.212" StartPoint="0.477,0.927">
```

```
                                <GradientStop Color="#FF000000"
Offset="0"/>
                                <GradientStop Color="#FF757474"
Offset="1"/>
                            </LinearGradientBrush>
                        </Button.BorderBrush>
                        <!--Red light-->
                        <Ellipse Name="elliRed" Style="{StaticResource
df}" >
                        </Ellipse>
                    </Button>
                    <Button Name="pbYellow" Background="#FF3C3C3C"
Foreground="#FF000000">
                        <Button.BorderBrush>
                            <LinearGradientBrush
EndPoint="1.219,0.212" StartPoint="0.477,0.927">
                                <GradientStop Color="#FF000000"
Offset="0"/>
                                <GradientStop Color="#FF757474"
Offset="1"/>
                            </LinearGradientBrush>
                        </Button.BorderBrush>
                        <!--Yellow light-->
                        <Ellipse Name="elliYellow"
Style="{StaticResource trafficLightYellow}" >

                        </Ellipse>
                    </Button>
                    <Button Name="pbGreen" Background="#FF3C3C3C"
Foreground="#FF000000">
                        <Button.BorderBrush>
                            <LinearGradientBrush
EndPoint="1.219,0.212" StartPoint="0.477,0.927">
                                <GradientStop Color="#FF000000"
Offset="0"/>
                                <GradientStop Color="#FF757474"
Offset="1"/>
                            </LinearGradientBrush>
                        </Button.BorderBrush>
                        <!--Green light-->
                        <Ellipse Name="elliGreen"
Style="{StaticResource trafficLightGreen}" >
                        </Ellipse>
                    </Button>
                </StackPanel>
            </Border>
            </GroupBox>
        </Grid>
    </Grid>
</UserControl>
```

UserControlLACT.xaml.vb

```
Private Sub timer_Tick(ByVal sender As Object, ByVal e As EventArgs)


        'Function check crane movement. e.g If crane is moving then
greenlight is on.
        Dim str As String
        Dim rstTemp As New ADODB.Recordset
        Dim i As Short
        Dim CraneNumber As Short

        For i = 0 To 3
            CraneNumber = i + 1
            str = "select request_to_stop, crane_fault " & "from crane
" & "where crane_number = '" & CraneNumber & "' "
            If i < 1 Then
                rstTemp.let_ActiveConnection(AdoCon)
            End If

            rstTemp.Open(str)
            'Screen.MousePointer = vbHourglass
            If IsCraneStopped(CraneNumber) = False Then '
                If rstTemp.Fields("crane_fault").Value = 1 Then
                    'Added new color values to the trafficlights
depend on crane's faults
                    trafficLightCompArray(i).elliRed.Fill = New
SolidColorBrush(Color.FromRgb(255, 0, 0))
                    trafficLightCompArray(i).elliYellow.Fill = New
SolidColorBrush(Color.FromRgb(53, 51, 0))
                    trafficLightCompArray(i).elliGreen.Fill = New
SolidColorBrush(Color.FromRgb(0, 255, 0))

                Else
                    'Added new color values to the trafficlights
depend on crane's stop request
                    If rstTemp.Fields("request_to_stop").Value = 1
Then
                        trafficLightCompArray(i).elliRed.Fill = New
SolidColorBrush(Color.FromRgb(53, 0, 0))
                        trafficLightCompArray(i).elliYellow.Fill = New
SolidColorBrush(Color.FromRgb(255, 255, 0))
                        trafficLightCompArray(i).elliGreen.Fill = New
SolidColorBrush(Color.FromRgb(0, 53, 0))

                    Else
                        trafficLightCompArray(i).elliRed.Fill = New
SolidColorBrush(Color.FromRgb(53, 0, 0))
                        trafficLightCompArray(i).elliYellow.Fill = New
SolidColorBrush(Color.FromRgb(53, 51, 0))
                        trafficLightCompArray(i).elliGreen.Fill = New
SolidColorBrush(Color.FromRgb(0, 255, 0))

                    End If
                End If
            Else ' crane has been stopped
```

```
                trafficLightCompArray(i).elliRed.Fill = New
SolidColorBrush(Color.FromRgb(255, 0, 0))
                trafficLightCompArray(i).elliYellow.Fill = New
SolidColorBrush(Color.FromRgb(53, 51, 0))
                trafficLightCompArray(i).elliGreen.Fill = New
SolidColorBrush(Color.FromRgb(0, 53, 0))

            End If
            rstTemp.Close()
        Next i

        mTimerStart = mTimerStart + 1
        'This increase mTimerStart value. If mTimerStart is more than
3000 then Application shutdown
        If mTimerStart > 3000 Then

            Application.Current.Shutdown()
        End If

End                                                                 Sub
```

Paperirullan malli

UserControlRollShape.xaml

```
<UserControl x:Class="UserControlRollShape"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Width="auto" Height="auto"
    >
    <UserControl.Resources>
    <!--ToolTip and ellipse styles-->
        <Style x:Key="EllipseStyle1" TargetType="{x:Type Ellipse}">
            <Setter Property="Fill">
                <Setter.Value>
                    <RadialGradientBrush>
                        <GradientStop Color="#FF000000" Offset="1"/>
                        <GradientStop Color="#FFD8C9C9" Offset="0"/>
                    </RadialGradientBrush>
                </Setter.Value>
            </Setter>
        </Style>
        <Style x:Key="MyTooltip2" TargetType="{x:Type ToolTip}">
            <Setter Property="SnapsToDevicePixels" Value="True"/>
            <Setter Property="Template">
                <Setter.Value>
                    <ControlTemplate TargetType="{x:Type ToolTip}">
                        <Border BorderBrush="#111111"
                          BorderThickness="1"
                          SnapsToDevicePixels="True">
                            <Border Width="Auto" Height="Auto"
Name="windowFrame"
                          BorderBrush="#999999"
                          BorderThickness="1"
                          SnapsToDevicePixels="True"
                          CornerRadius="0"
                          Background="WhiteSmoke"
TextBlock.Foreground="Black"
                        TextBlock.FontFamily="Verdana"
TextBlock.FontSize="11"
                        TextBlock.FontWeight="Bold"
                        Margin="0" Padding="3"
                          >
                                  <ContentPresenter />
                              </Border>
                        </Border>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>
        </Style>
    </UserControl.Resources>
    <Viewbox>
        <Viewbox.ToolTip>
        <!--ToolTip shows location of paper-roll-->
            <ToolTip Style="{StaticResource MyTooltip2}"
x:Name="toolTipRollShape">
```

```
                <ToolTip.BitmapEffect>
                    <BitmapEffectGroup>
                        <DropShadowBitmapEffect Color="Black"
Direction="120" ShadowDepth="25" Softness="1" Opacity="0.5"/>
                    </BitmapEffectGroup>
                </ToolTip.BitmapEffect>
            </ToolTip>
        </Viewbox.ToolTip>
            <Grid Background="Transparent">
            <!--Paper-roll location shape-->
            <Ellipse Width="20" Height="20" Name="elli"
Stroke="#FF000000" StrokeThickness="3" Fill="#FFADADAD">
            </Ellipse>
            <!--Paper-roll amount-->
            <Label Margin="2,0,0,0" x:Name="lblShapeRoll">10</Label>
    </Grid>
    </Viewbox>
</UserControl>
```

Paperirullapaikan näkyvyystoiminto

UserControlLACT.xaml.vb

```
Private Sub updateRollLocationBorder(ByRef rollobject As
UserControlRollShape, ByRef storage As Short, ByRef rstTmp As
ADODB.Recordset, ByRef rollAmount As Short)

    'Function set color attributes to RollShape
    If storage = 1 Then

        If rollAmount > 0 Then

             Dim mySolidColorBrush As New SolidColorBrush()
             mySolidColorBrush.Color = Color.FromArgb(240, 255, 255,
255)
             rollobject.elli.Fill = mySolidColorBrush

        End If
        If rstTmp.Fields("cfs").Value > 0 Then
                rollobject.elli.Stroke = Brushes.Brown
        End If
        If rstTmp.Fields("cfs").Value > 0 And
rstTmp.Fields("cfr").Value > 0 Then
            rollobject.elli.Stroke = Brushes.Magenta
        End If

        If rstTmp.Fields("cfr").Value > 0 And
rstTmp.Fields("cfs").Value = 0 Then
            rollobject.elli.Stroke = Brushes.Orange
        End If

        If cbShowRollPileOnly.IsChecked = True Then
            If rollAmount > 0 Then
            rollobject.Opacity = 1.0
            Else
            'Opacity is low if checkbox (cbShowRollPileOnly) is
checked
            rollobject.Opacity = 0.1
        End If
        End If
    End If

    If storage = 2 Then
        If rollAmount > 0 Then
             Dim mySolidColorBrush As New SolidColorBrush()
             mySolidColorBrush.Color = Color.FromArgb(240, 255, 255,
255)
             rollobject.elli.Fill = mySolidColorBrush
        End If
        If rstTmp.Fields("cfs").Value > 0 Then
                rollobject.elli.Stroke = Brushes.Brown
        End If
    If rstTmp.Fields("cfs").Value > 0 And rstTmp.Fields("cfr").Value >
0 Then
            rollobject.elli.Stroke = Brushes.Magenta
```

```
            End If
            If rstTmp.Fields("cfr").Value > 0 Then
                rollobject.elli.Stroke = Brushes.Orange
            End If
            If cbShowRollPileOnly.IsChecked = True Then
                If rollAmount > 0 Then
                rollobject.Opacity = 1.0
                Else
                rollobject.Opacity = 0.1
            End If
            End If
        End If

End Sub
```

Paperirullapinon sivuttaisnäkymä

UserControlLACT.xaml.vb

```
Private Sub sideViewRollInfoOnMouse(ByVal sender As
UserControlRollPileShape, _
      ByVal e As EventArgs)

        'Clear all data in ListView
        sender.rollDataListView.Items.Clear()
        'Roll's description of attributes
        Dim sqlQuery1 As String = ""
        Dim sqlQuery2 As String = ""
        Dim rollNr As String = "Roll "
        Dim diameter As String = "Diameter: "
        Dim width As String = "Width: "
        Dim length As String = "Length: "
        Dim age As String = "Created: "
        Dim grade As String = "Grade: "
        Dim grammage As String = "Grammage: "
        Dim core As String = "Core: "
        Dim diameter2 As String = "Diameter: "
        Dim altitude As String = "Altitude: "
        Dim finishingCode As String = "Finishing Code: "
        Dim position As String = "Position: "
        Dim statusCode As String = "Status: "
        Dim weight As String = "Weight: "
        Dim rType As String = "Type: "
        Dim rstTemp As New ADODB.Recordset
        Dim myDA As New OleDbDataAdapter
        Dim ds As New DataSet

        Dim data As String = "There's no data in roll!!!"
            sqlQuery1 = "SELECT r.ROLL_NUMBER as RollNumber, " &
"r.DIAMETER as Diameter, " & "t.nom_width as Width, " & "r.length,
r.AGE, t.GRADEID , t.basis_weight, r.core_size, " & "t.nom_diameter, "
& "t.finishing_code, r.Position, r.status_code, " & "r.ALTITUDE,
r.weight, " & "r.type "
            sqlQuery2 = "FROM ROLLS r, roll_type t " & "WHERE LOCATION
= '" & sender.rollPileElli.Tag & "' and position = " & 1 & " " & "and
r.Type = t.Type " & "ORDER BY ALTITUDE DESC"
            rstTemp.let_ActiveConnection(AdoCon)
            rstTemp.Open(sqlQuery1 + sqlQuery2)

            myDA.Fill(ds, rstTemp, "MyTable")

            If dgTblPileData.Items.Count = 0 Then
            'No data in roll then error message
                sender.rollDataListView.Items.Add(data)

                Else
                'Add values to variables
                rollNr +=
ds.Tables(0).Rows(sender.LayoutRoot.Tag).Item(0).ToString
                diameter +=
ds.Tables(0).Rows(sender.LayoutRoot.Tag).Item(1).ToString
```

```
                width +=
ds.Tables(0).Rows(sender.LayoutRoot.Tag).Item(2).ToString
                'length +=
ds.Tables(0).Rows(sender.LayoutRoot.Tag).Item(3).ToString
                age +=
ds.Tables(0).Rows(sender.LayoutRoot.Tag).Item(4)
                grade +=
ds.Tables(0).Rows(sender.LayoutRoot.Tag).Item(5).ToString
                grammage +=
ds.Tables(0).Rows(sender.LayoutRoot.Tag).Item(6).ToString
                core +=
ds.Tables(0).Rows(sender.LayoutRoot.Tag).Item(7).ToString
                diameter2 +=
ds.Tables(0).Rows(sender.LayoutRoot.Tag).Item(8).ToString
                finishingCode +=
ds.Tables(0).Rows(sender.LayoutRoot.Tag).Item(9).ToString
                position +=
ds.Tables(0).Rows(sender.LayoutRoot.Tag).Item(10).ToString
                statusCode +=
ds.Tables(0).Rows(sender.LayoutRoot.Tag).Item(11).ToString
                altitude +=
ds.Tables(0).Rows(sender.LayoutRoot.Tag).Item(12).ToString
                weight +=
ds.Tables(0).Rows(sender.LayoutRoot.Tag).Item(13).ToString
                rType +=
ds.Tables(0).Rows(sender.LayoutRoot.Tag).Item(14).ToString
                'Add variables to sender's ListView
                sender.rollDataGB.Header = rollNr
                sender.rollDataListView.Items.Add(diameter)
                sender.rollDataListView.Items.Add(width)
                sender.rollDataListView.Items.Add(age)
                sender.rollDataListView.Items.Add(grade)
                sender.rollDataListView.Items.Add(grammage)
                sender.rollDataListView.Items.Add(core)
                sender.rollDataListView.Items.Add(diameter2)
                sender.rollDataListView.Items.Add(finishingCode)
                sender.rollDataListView.Items.Add(position)
                sender.rollDataListView.Items.Add(statusCode)
                sender.rollDataListView.Items.Add(altitude)
                sender.rollDataListView.Items.Add(weight)
                sender.rollDataListView.Items.Add(rType)


            End If
            'If location has not rolls then visibility of tooltip is
hidden
        If sender.LayoutRoot.Tag.ToString = "" Then
            sender.rollPileToolTip.Visibility =
Windows.Visibility.Hidden
            Else
            sender.rollPileToolTip.Visibility =
Windows.Visibility.Visible
            sender.rollPileToolTip.Content = sender.rollDataGB
        End If

    End Sub
```

```
Private Sub sideViewRollInformation(ByVal sLocation As String, ByVal
rollcount As Short)
    'If grid has content then clear
    If grRollPileInfo.Children.Count <> 0 Or
grRollPileInfoShadow.Children.Count <> 0 Then
        grRollPileInfo.Children.Clear()
        grRollPileInfoShadow.Children.Clear()
    End If
    'Arrays of UserControlRollPileShape
    Dim shapePile(rollcount - 1) As UserControlRollPileShape

    If rollcount > 0 Then
    'Paper-roll stacking
    '1st Shadow stack
         For g = 0 To shapePile.Count - 1
            shapePile(g) = New UserControlRollPileShape
            shapePile(g).Margin = New Thickness(0, g * (-75), 0, 0)
            shapePile(g).Opacity = 0.1
            grRollPileInfoShadow.Children.Add(shapePile(g))

        Next
    '2nd Shadow stack
         For g = 0 To shapePile.Count - 1
            shapePile(g) = New UserControlRollPileShape
            shapePile(g).Margin = New Thickness(100, g * (-75), 0, 0)
            shapePile(g).Opacity = 0.1
            grRollPileInfoShadow.Children.Add(shapePile(g))

        Next
        grRollPileInfo.Margin = New Thickness(47, 33, 0, 0)
    'Real stack
        For o = 0 To shapePile.Count - 1
            shapePile(o) = New UserControlRollPileShape
            shapePile(o).Margin = New Thickness(0, o * (-75), 0, 0)
            shapePile(o).LayoutRoot.Tag = rollcount - 1
            shapePile(o).rollPileElli.Tag = sLocation
            AddHandler shapePile(o).MouseEnter, AddressOf
sideViewRollInfoOnMouse
            grRollPileInfo.Children.Add(shapePile(o))
            rollcount = rollcount - 1

        Next
    'Grid's height depending on stack height
        grRollPileInfo.Height = shapePile.Count * 80
        grRollPileInfoShadow.Height = shapePile.Count * 80
    End If

End Sub
```

UserControlRollPileShape.xaml

```
<UserControl
           xmlns="http://schemas.microsoft.com/winfx/2006/xaml/present
ation"
           xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
           xmlns:d="http://schemas.microsoft.com/expression/blend/2008
"
           xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
           xmlns:styles="clr-namespace:APP_Hainan"
           mc:Ignorable="d"
           x:Class="UserControlRollPileShape"
           x:Name="UserControl"
           xmlns:dg="http://schemas.microsoft.com/wpf/2008/toolkit">
    <UserControl.Resources>
        <!--Tooltip style-->
        <Style x:Key="MyTooltip2" TargetType="{x:Type ToolTip}">
            <Setter Property="SnapsToDevicePixels" Value="True"/>
            <Setter Property="Template">
                <Setter.Value>
                    <ControlTemplate TargetType="{x:Type ToolTip}">
                        <Border BorderBrush="#111111"
                          BorderThickness="1"
                          SnapsToDevicePixels="True"
                          Margin="25,0,0,0">
                            <Border Width="Auto" Height="Auto"
Name="windowFrame"
                          BorderBrush="#999999"
                          BorderThickness="1"
                          SnapsToDevicePixels="True"
                          CornerRadius="0"
                          Background="WhiteSmoke"
TextBlock.Foreground="Black"
                        TextBlock.FontFamily="Verdana"
TextBlock.FontSize="11"
                        TextBlock.FontWeight="Bold"
                        Margin="0" Padding="3"
                          >
                                  <ContentPresenter />
                            </Border>
                        </Border>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>
        </Style>
        <!--Animation start when mouse is over the roll-->
        <Storyboard x:Key="strokeThickness">
           <DoubleAnimationUsingKeyFrames BeginTime="00:00:00"
Storyboard.TargetName="path"
Storyboard.TargetProperty="(Shape.StrokeThickness)">
                    <SplineDoubleKeyFrame KeyTime="00:00:00"
Value="3"/>
                    <SplineDoubleKeyFrame KeyTime="00:00:00.3000000"
Value="4"/>
           </DoubleAnimationUsingKeyFrames>
```

```
            <DoubleAnimationUsingKeyFrames BeginTime="00:00:00"
Storyboard.TargetName="rollPileElli"
Storyboard.TargetProperty="(Shape.StrokeThickness)">
                        <SplineDoubleKeyFrame KeyTime="00:00:00"
Value="3"/>
                        <SplineDoubleKeyFrame KeyTime="00:00:00.3000000"
Value="4"/>
            </DoubleAnimationUsingKeyFrames>
        </Storyboard>
        <Storyboard x:Key="strokeThickness2">
            <DoubleAnimationUsingKeyFrames BeginTime="00:00:00"
Storyboard.TargetName="path"
Storyboard.TargetProperty="(Shape.StrokeThickness)">
                        <SplineDoubleKeyFrame KeyTime="00:00:00"
Value="4"/>
                        <SplineDoubleKeyFrame KeyTime="00:00:00.3000000"
Value="3"/>
            </DoubleAnimationUsingKeyFrames>
            <DoubleAnimationUsingKeyFrames BeginTime="00:00:00"
Storyboard.TargetName="rollPileElli"
Storyboard.TargetProperty="(Shape.StrokeThickness)">
                        <SplineDoubleKeyFrame KeyTime="00:00:00"
Value="4"/>
                        <SplineDoubleKeyFrame KeyTime="00:00:00.3000000"
Value="3"/>
            </DoubleAnimationUsingKeyFrames>
        </Storyboard>

    </UserControl.Resources>
            <UserControl.Triggers>
                        <EventTrigger RoutedEvent="Mouse.MouseEnter">
                                    <BeginStoryboard
Storyboard="{StaticResource strokeThickness}"
x:Name="strokeThickness2_BeginStoryboard"/>
                        </EventTrigger>
                        <EventTrigger RoutedEvent="Mouse.MouseLeave">
                                    <BeginStoryboard
Storyboard="{StaticResource strokeThickness2}"
x:Name="strokeThickness2_BeginStoryboard1"/>
                        </EventTrigger>
            </UserControl.Triggers>
    <Viewbox Width="50" Height="60" OpacityMask="{x:Null}">
        <Viewbox.ToolTip>
            <ToolTip Visibility="Hidden" Style="{StaticResource
MyTooltip2}" x:Name="rollPileToolTip">
                <GroupBox Background="Transparent" x:Name="rollDataGB"
Width="auto" Height="auto">
                    <ListView Background="Transparent"
x:Name="rollDataListView"></ListView>
                </GroupBox>
            </ToolTip>
        </Viewbox.ToolTip>
        <!--Rollpile shape-->
        <Grid x:Name="LayoutRoot" Width="50" Height="60"
Background="Transparent">
            <Path Margin="0,9.295,0,0" Stretch="Fill"
Stroke="#FF000000" StrokeThickness="3" Data="M81,148 C81,148
82.262809,420.15369 86.5,430.5 100.83369,465.4998 168.16682,503.48743
```

```
278.83352,504.16658 387.50029,504.83346 474.16737,474.83313
495.5,447.50013 510.29193,428.54758 503.5,146.5 503.5,146.5 z"
d:LayoutOverrides="HorizontalAlignment, VerticalAlignment"
x:Name="path">
                                        <Path.Fill>

          <LinearGradientBrush EndPoint="0.492,0.315"
StartPoint="0.495,1.074">

          <GradientStop Color="#FF666666" Offset="0"/>

          <GradientStop Color="#FFFFFFFF" Offset="1"/>

          </LinearGradientBrush>
                                        </Path.Fill>
                                </Path>

                                <Ellipse x:Name="rollPileElli"
Margin="0.143,0" VerticalAlignment="Top" Height="21.021"
Stroke="#FF000000" StrokeThickness="3"
d:LayoutOverrides="HorizontalAlignment, VerticalAlignment"
RenderTransformOrigin="0.5,0.5">
                                        <Ellipse.RenderTransform>
                                                <TransformGroup>

          <ScaleTransform ScaleX="-1" ScaleY="1"/>

          <SkewTransform AngleX="0" AngleY="0"/>

          <RotateTransform Angle="0"/>

          <TranslateTransform X="0" Y="0"/>

          </TransformGroup>
                                        </Ellipse.RenderTransform>
                                        <Ellipse.Fill>

          <RadialGradientBrush GradientOrigin="0.5,0.5">

          <RadialGradientBrush.RelativeTransform>

          <TransformGroup>

                    <ScaleTransform CenterX="0.5" CenterY="0.5"
ScaleX="1.286" ScaleY="1.286"/>

                    <SkewTransform AngleX="0" AngleY="0"
CenterX="0.5" CenterY="0.5"/>

                    <RotateTransform Angle="0" CenterX="0.5"
CenterY="0.5"/>

                    <TranslateTransform X="0" Y="0"/>

          </TransformGroup>

          </RadialGradientBrush.RelativeTransform>
```

```
            <GradientStop Color="#FFADADAD" Offset="1"/>

            <GradientStop Color="#FFFFFFFF" Offset="0"/>

            <GradientStop Color="#FFDCDCDC" Offset="0.471"/>

            </RadialGradientBrush>
                                          </Ellipse.Fill>
                                </Ellipse>
                    </Grid>
                    </Viewbox>
</UserControl>
```

Twip -ja pikselimuunnokset

TwipToPixels:

```
Function TwipsToPixels(ByVal iTwips As Integer) As Integer
        Dim twipHelp As Integer

        twipHelp = iTwips * (0.066666667)

        Return twipHelp

End Function
```

PixelsToTwips:

```
Function PixelsToTwips(ByVal iPixels As Integer) As Integer
        Dim twipHelp As Integer

        twipHelp = iPixels * 15

        Return twipHelp

End Function
```

VB.NET-kontrollien indeksointi lomakkeessa

frmLACT.Designer.vb (Indeksin asettelu boldattu):

```
'
'LabelXindex
'
Me.LabelXindex.AutoSize = True
Me.LabelXindex.BackColor = System.Drawing.Color.Transparent
Me.lblArrXIndex.SetIndex(Me.LabelXindex, CType(0, Short))
Me.LabelXindex.Location = New System.Drawing.Point(-3, 25)
Me.LabelXindex.Name = "LabelXindex"
Me.LabelXindex.Size = New System.Drawing.Size(13, 13)
Me.LabelXindex.TabIndex = 4
Me.LabelXindex.Text = "1"
```

Tietokantayhteys ADO.NET-rajapintaa käyttäen

HainanDataClasses.designer.vb:

```
Public ReadOnly Property ERRORLOGs() As System.Data.Linq.Table(Of
ERRORLOG)
                        Get
                                    Return Me.GetTable(Of ERRORLOG)
                        End Get
            End Property

            Public ReadOnly Property PLC_ERROR_BITMAPs() As
System.Data.Linq.Table(Of PLC_ERROR_BITMAP)
                        Get
                                    Return Me.GetTable(Of
PLC_ERROR_BITMAP)
                        End Get
            End Property

            Public ReadOnly Property languages() As
System.Data.Linq.Table(Of language)
                        Get
                                    Return Me.GetTable(Of language)
                        End Get
            End Property

            Public ReadOnly Property NEW_ROLLs() As
System.Data.Linq.Table(Of NEW_ROLL)
                        Get
                                    Return Me.GetTable(Of NEW_ROLL)
                        End Get
            End Property

            Public ReadOnly Property ROLL_TYPEs() As
System.Data.Linq.Table(Of ROLL_TYPE)
                        Get
                                    Return Me.GetTable(Of ROLL_TYPE)
                        End Get
            End Property
End Class
```

```
<Column(Storage:="_CRANE", DbType:="NVarChar(255)")>  _
            Public Property CRANE() As String
                        Get
                                    Return Me._CRANE
                        End Get
                        Set
                                    If (String.Equals(Me._CRANE, value) =
false) Then
                                                Me._CRANE = value
                                    End If
                        End Set
```

```
            End Property

            <Column(Storage:="_DESCRIPTION", DbType:="NVarChar(255)")>
_
            Public Property DESCRIPTION() As String
                        Get
                                    Return Me._DESCRIPTION
                        End Get
                        Set
                                    If (String.Equals(Me._DESCRIPTION,
value) = false) Then
                                                Me._DESCRIPTION = value
                                    End If
                        End Set
            End Property
            <Column(Storage:="_SENDER", DbType:="NVarChar(255)")>  _
            Public Property SENDER() As String
                        Get
                                    Return Me._SENDER
                        End Get
                        Set
                                    If (String.Equals(Me._SENDER, value) =
false) Then
                                                Me._SENDER = value
                                    End If
                        End Set
            End Property

            <Column(Storage:="_TIME_STAMP", DbType:="DateTime")>  _
            Public Property TIME_STAMP() As System.Nullable(Of Date)
                        Get
                                    Return Me._TIME_STAMP
                        End Get
                        Set
                                    If (Me._TIME_STAMP.Equals(value) =
false) Then
                                                Me._TIME_STAMP = value
                                    End If
                        End Set
            End Property
```